

# Mastering ArchiMate

Edition I

By Gerben Wierda

A Serious Introduction to  
the ArchiMate® Enterprise  
Architecture Modeling  
Language

Copyright © 2012 by Gerben Wierda, The Netherlands

First Edition: 15 October 2012. Fourth Printing: 5 January 2013. This is the Screen Edition.

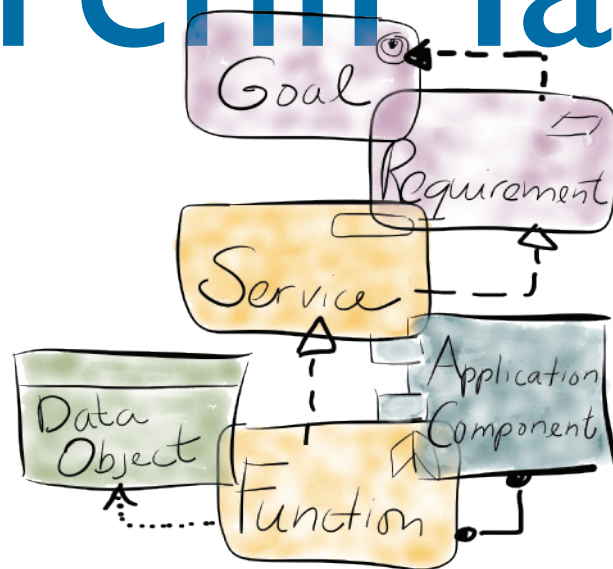
Neither the author, not the publisher, nor the printer accept any responsibility or liability for loss or damage occasioned to any person or property through using the material, instructions, methods or ideas contained in or distributed with this book, or acting or refraining from acting as a result of such use. The author and publisher and printer expressly disclaim all implied warranties, including merchantability of fitness for any particular purpose.

ISBN 978-90-819840-0-3

License & Print Info: see "License" on page 12

Contact info: [info@masteringarchimate.com](mailto:info@masteringarchimate.com)

# Mastering ArchiMate



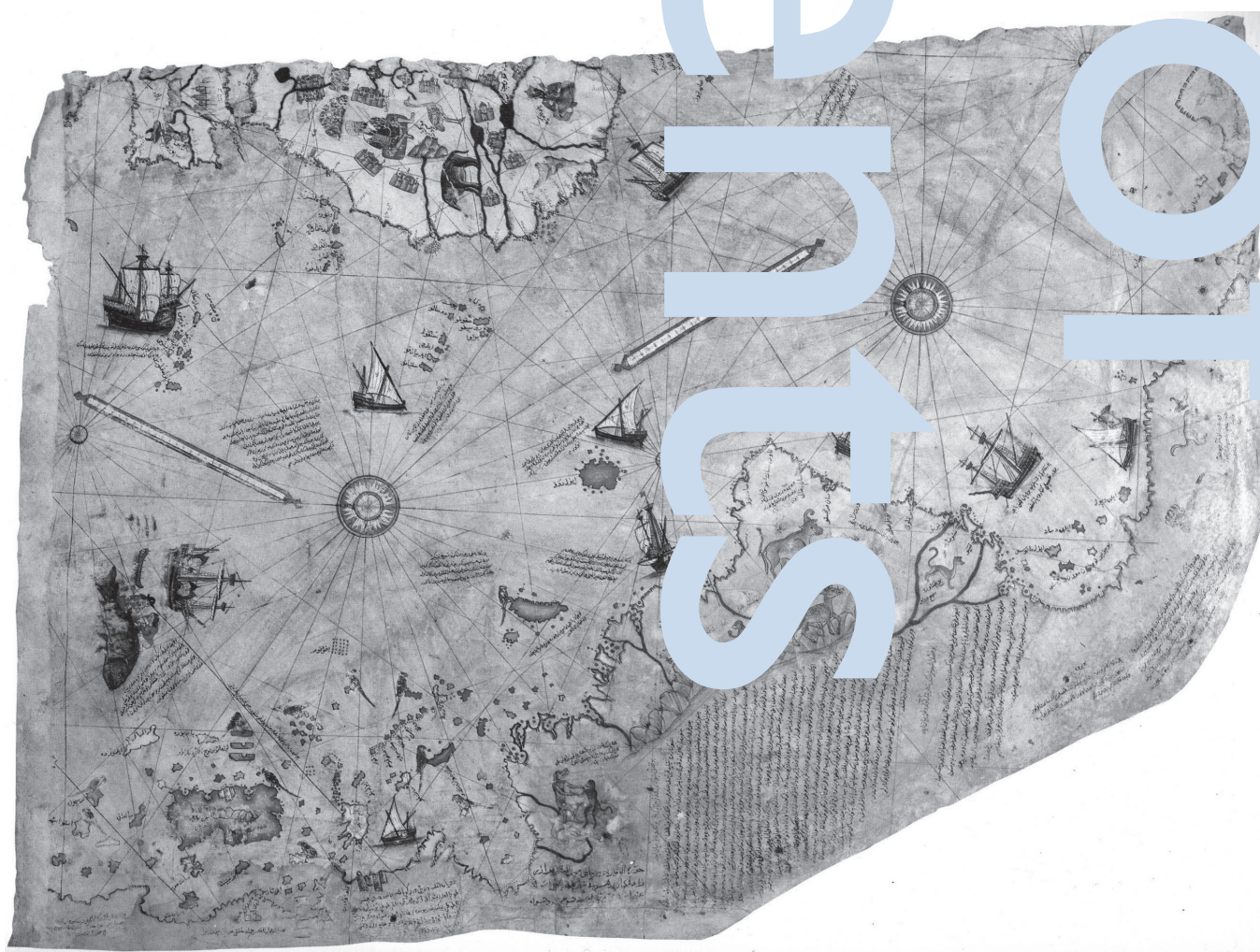
Edition I

ISBN 978-90-819840-0-3 (Screen PDF)

ISBN 978-90-819840-1-0 (Bound print)



# Table of Contents





# Table of Contents

## Introduction .....9

1. Why This Book? .....	9
1.1 Uncle Ludwig.....	9
2. Enterprise Architecture.....	10
2.1 Where are the principles and guidelines? .....	11
2.2 Disclaimer.....	11
3. Gratitude .....	11
4. License .....	12
5. Release Notes for Edition 1 .....	12

## ArchiMate Basics ..... 15

1. Objects and Relations: 3x3x3 .....	15
1.1 Applications and Business .....	15
1.2 The double use of Used-By .....	17
1.3 Business Function.....	18
1.4 Business Actor .....	18
1.5 Adding Technical Infrastructure to the Mix.....	18
1.6 System Software and Devices .....	19
1.7 Composition and Aggregation .....	20
1.8 Nesting.....	20
1.9 Using a Node to encapsulate infrastructure .....	21
1.10 Events, Triggers and Flows .....	22
1.11 The (almost) complete Picture.....	22
2. Other Objects and Relations.....	23
2.1 Collaborations and Interactions .....	23
2.2 The Association Relation.....	24
2.3 The Specialization Relation.....	24
2.4 Products, Contracts and Value .....	25
2.5 Representations and Meanings .....	25
2.6 Network and Communication Path.....	26
2.7 Automated Processes .....	26
2.8 The Grouping Relation.....	26
2.9 The Junction Relation.....	27
2.10 Location .....	27
2.11 The Complete Picture .....	27

2.12 Closing Remark .....	27
3. Derived Relations .....	29
3.1 Derived Structural Relations .....	29
3.2 Derived Dynamic Relations.....	30
4. Beginner's Pitfalls .....	30
4.1 Pitfalls of Derived Relations.....	30
4.2 Don't trust the language or the tool blindly: know what you are doing .....	31
4.3 Misunderstanding the standard meta-model diagram.....	32
5. Some Noteworthy Aspects of ArchiMate ....	32
5.1 Introduction.....	32
5.2 Separation of Actor and Behavior .....	33
5.3 One View Type .....	33
5.4 On the direction of structural relations under the assumption of 'worst case' .....	34
5.5 On the limitations of derived relations .....	34
5.6 Why two types of software? .....	35
5.7 There is more than one-to-one .....	36

## Style & Patterns ..... 39

6. Aesthetics (Style).....	39
6.1 Why aesthetics matters .....	39
6.2 Arranging relations .....	39
6.3 Sizing and Arranging objects .....	41
6.4 Using color.....	42
6.5 Grouping.....	43
6.6 About labels .....	43
6.7 Don't Use the 'Children's Forms' .....	44
7. Patterns.....	44
7.1 The Use of Patterns .....	44
7.2 A Basic TI Pattern: A Database .....	44
7.3 Modeling Spreadsheet Applications.....	46
7.4 Modeling an Internet Browser.....	48
7.5 More 'application platforms' .....	49
7.6 Infrastructure 'Building Blocks' .....	50
7.7 Application Deployment Patterns.....	51

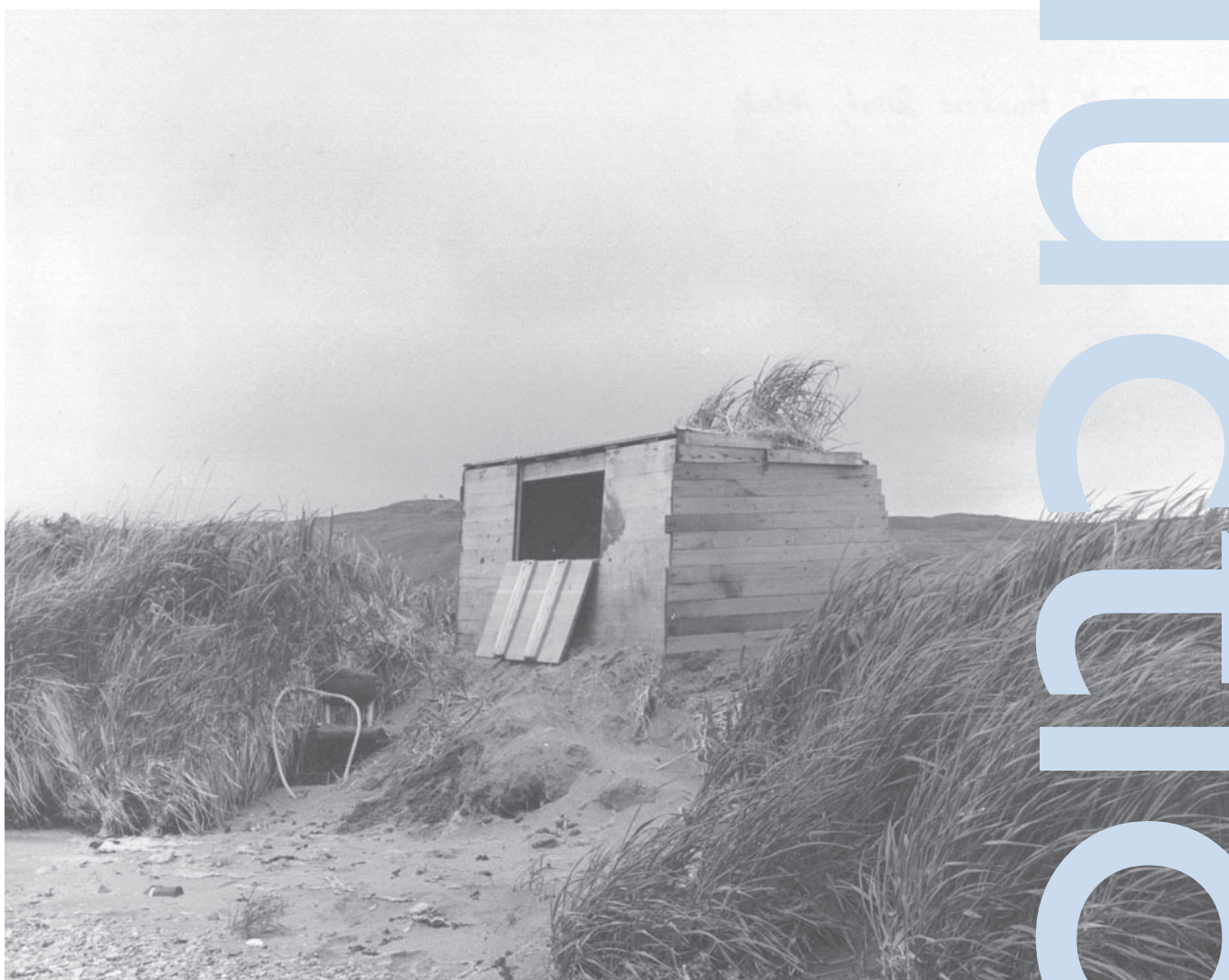
7.8 Deployment Pattern: Standalone PC Application ...	51
7.9 Deployment Pattern: Standalone PC Application with File Server based Data .....	51
7.10 Deployment Pattern: Classic Two-Tier Client-Server Application.....	52
7.11 Deployment Pattern: Two-Tier Client-Server Application with mounted Client Application and two databases	52
7.12 Deployment Pattern: Two-Tier Client-Server with a Remote Server (ASP) .....	53
7.13 Deployment Pattern: Three-Tier Application.....	55
7.14 Deployment Pattern: Software as a Service (SaaS)	55
7.15 We only model what we can change.....	56
7.16 Deployment Pattern: Providing a local ASP.....	56
7.17 The Use of Patterns (reprise).....	57
7.18 Infrastructure Pattern: Database Replication.....	57
7.19 Infrastructure Pattern: High-Available Database Cluster.....	60
7.20 Infrastructure Pattern: High-Available Server .....	60
7.21 Using Collections and Abstraction in a model ....	60
7.22 Concurrent Realization? .....	61
8. Anti-patterns.....	63
8.1 Multiple 'Realizers' for a single Service .....	63
8.2 Application Collaboration is an Anthropomorphism	63
8.3 Using the Association Relation .....	65
8.4 Using properties too much .....	65
9. An Example from the Real World.....	65
<b>Advanced Patterns.....</b>	<b>71</b>
10. Business Function and Business Process ..	71
10.1 On ArchiMate's divide between behaviour and structure.....	71
10.2 Business Function or Business Process? .....	72
10.3 Good Old Fashioned Business Function .....	74
10.4 The 'End-to-end' Business Process .....	75
10.5 Business Proces Modeling versus Business Layer Architecture.....	76
11. How ArchiMate Helps Choosing Your Business Functions .....	77
11.1 Dividing your enterprise's behavior into Business Functions.....	77
11.2 Concurrent Functional Landscapes .....	81
12. Secondary and Tertiary Architecture.....	84
12.1 Introduction.....	84
12.2 Primary Architecture.....	84
12.3 Secondary Architecture: Development.....	84
12.4 Intermezzo .....	85
12.5 Secondary Architecture: System Exploitation .....	85
12.6 Secondary Architecture: Application Maintenance	86
12.7 Tertiary Architecture: Application Owner .....	86
12.8 Tertiary Architecture: other roles .....	86

12.9 Making it Practical: shortcuts.....	87
13. Modeling Risk & Security .....	90
13.1 Security Architecture .....	90
13.2 Risk .....	90
13.3 ArchiMate 2.0 Motivation Extension .....	91
13.4 Modeling Risks in ArchiMate 2.0 .....	92
14. More Platform Thinking .....	94
14.1 Low Level Data Entry.....	94
<b>Model Use &amp; Maintenance.....</b>	<b>97</b>
15. Construction & Use Views .....	97
16. Model Maintenance .....	101
<b>Discussing ArchiMate .....</b>	<b>103</b>
17. Proposed Improvements .....	103
17.1 Switching Strength of Assignment and Realization	103
17.2 Service as Composite Part of Function/Process...	103
17.3 Automated Processes .....	104
17.4 Summary of the previous three change proposals	106
17.5 Have only Business Service in the Product Aggregate?	107
17.6 Actors Aggregate Roles?.....	108
17.7 Only use Assignment for performed-by.....	108
17.8 Allow multiple parents in a Composition .....	109
17.9 Infrastructure Collaboration and Interaction to model Clustering.....	110
17.10 Clean Up the Location Concept, possibly change Grouping.....	110
17.11 Remove Nesting as a Language Concept .....	110
17.12 Make the Access relation bidirectional .....	111
17.13 Get rid of Interactions.....	111
17.14 Get rid of unwanted derived relations explicitly	111
17.15 Get rid of Meaning .....	112
17.16 Make a Motivational Element in ArchiMate's Motivation Extension a Specialization of Business Object	112
17.17 Give Product a non-passive status.....	112
17.18 Improve the description of Business Function ..	112
17.19 Are the proposed improvements really necessary? .	112
<b>Tooling.....</b>	<b>115</b>
18. Multiple Models of One Reality.....	115
19. Tool Aspects that may be important .....	116
20. Tool Reviews .....	116
<b>Index .....</b>	<b>119</b>
<b>List of Figures.....</b>	<b>123</b>
<b>Finger .....</b>	<b>129</b>



This page intentionally not left blank

# Introductory





# Introduction

## I. Why This Book?

This book grew out of the desire to share the things I learned over the course of several years with respect to seriously employing the ArchiMate® Enterprise Architecture Modeling language. As Lead Architect, working for the Asset Management unit of the largest Fiduciary Manager in the world (>B€ 300 Assets under Management as I write this), I introduced the use of ArchiMate because it seemed to me — for a variety of reasons — the only reasonable choice for our modeling in the line of our Enterprise Architecture work. The choice was based on an estimate of the practicality of the language, but I am satisfied with how it works in our daily practice. This practice includes the maintenance of a single very large (tens of thousands of objects and relations) ‘current state’ model of our enterprise, built along strict guidelines and used for analysis, reporting and as source for other systems (e.g. the CMDB of Infrastructure Management). The scope and detail of our modeling has taught us very valuable lessons about ArchiMate modeling, which I am sharing in this book.

I decided at the start of our ArchiMate use to stick almost religiously to the official definitions and not think or talk about diverging until we had a reasonable body of experience. Only when you have enough experience are you capable of really estimating the effect of the choices you have when changing the language. We still stay very close to what ArchiMate prescribes to this day.

This book also grew out of the desire to provide a better introduction than what was available. In my experience, what is available is often pretty limited in scope and in my opinion sometimes even damaging in its explanations as you will not learn the things to become a *good* modeler in ArchiMate but you will learn some random patterns without learning the pitfalls of those patterns. No introduction I have seen actually explains the language well enough and I have not seen introductions that actually prepare you for sizable modeling work.

The best document until now (if you ask me) is the original paper from (then) the Telematica Instituut (Telin, now called Novay) of the Technical University Twente in The Netherlands, describing the pre-1.0 original ArchiMate standard. This comes close to the official specification — which itself is of course by definition right — but not

automatically the best way to get educated. This book is an attempt at something better.

I intend to do two things:

- Give a decent initial introduction in the concepts and relations that make up the language;
- Present a number of patterns and uses that may be useful when modeling in the language

### 1.1 Uncle Ludwig

Here and there, you will find references to ‘Uncle Ludwig’. Uncle Ludwig stands for the twentieth century philosopher Ludwig Wittgenstein, who some characterize as the most important analytic philosopher (as opposed to, say, moral philosopher) to date. Wittgenstein in his entire philosophical life concentrated on ‘meaning’. His result came in two parts. The first part in his youth, where he tried to build meaning on top of logic. But the results were limited (but ironically had the most influence of all of his work in the computer science discipline). Later in life he tried to answer what meaning then was for all that rest of what we say, where logic does not give you the definitive answer. He came up with the solution ‘meaning lies hidden in correct use’. Given that ArchiMate is in part a service- (and thus use-) oriented language, this maxim is useful here and there. Actually, I find the maxim extremely useful for work even beyond modeling, as wondering about the actual use of documents sheds light about their meaning.

If you want to know more about Wittgenstein, I suggest “Wittgenstein’s Place in 20th Century Analytical Philosophy” by P.M.S. Hacker. Wittgenstein has been misinterpreted by many (e.g. when people misinterpret him as having stated that “meaning is *equivalent* to use”). Hacker not only understands Wittgenstein (he has written extensive and insightful analyses) but he also can explain it rather well. Wittgenstein often sounds daunting to people, but in my experience it can be pretty practical.

## 2. Enterprise Architecture

So you're working in an organization. The organization consists of many people, organized in organizational structure, in groups, departments and sections, and — if large enough — business units or even separate companies. If you look at what these people do, you look at business functions — say, 'after-sales' — or business processes — say 'handling a warranty claim from a customer'. These business functions and processes are a way to look at the behavior of your organization.

Now these days you have computers to support your work. You might not look at them anymore as computers (e.g. the iPhone or iPad you are reading this book on), but they are. And if your work is not shoe repair, plumbing, building, etc., your work will involve handling information. And even if it is shoe repair, the informational aspects of your work (planning, billing, accounting, etc.) are supported by IT. You *use* IT in the line of your work. The IT that supports people also has structure and behavior, just like the business itself. You use *applications*, and these 'running' applications have *functions* (i.e. behavior) and deliver *services* (support) to the business process. The applications themselves need an IT infrastructure to 'run', maybe large servers, maybe just your laptop or iPhone (where the applications are called 'Apps'), and there have to be all kinds of networks so all these systems can communicate with each other.

For about half a century now, the information revolution has moved most data from paper and other non-electronic media to IT systems, even the data that eventually is printed on paper. A *unimedia* revolution has taken place: from all kinds of different storage and transport media, the information has been digitized and been moved to being small electrically charged or magnetized spots, each of these spots representing a single yes/no choice: a bit. We hardly think about that level, but we all know what a *file* is these days, and generally we do not think of the paper original that the term originally stood for.

In large organizations, all these applications and files that support the business have become an almost impossible to control, complex landscape, where many things can and do go wrong and where change is fraught with peril. Because changing something here will crash something somewhere else in a landscape that is one big web of dependencies of business and IT. And even if that was not the case, translating business strategy and requirements to the right IT-support, or using IT-innovations to improve your business are difficult. Because, contrary to popular belief and partly as a result of that inertia-building web of dependencies, IT does not change fast. Building a new office is often a process that takes less time than implementing (let alone building and/or implementing) a new core IT system.

This web of objects (bank accounts, bills, roles and actors, applications, data, servers, files, networks, etc.) and relations between them is what Enterprise Architecture is about. It is about the design of your business and the IT that supports it. It is about having the right business organization, having the right IT for your business and letting the business innovate on the basis of the possibilities of IT, now and in the future. Especially, it is meant to lead

to better IT choices, because, as stated above, IT is often more difficult to change than the business.

The appearance of Enterprise Architects in this field is relatively recent. Not too long ago, if you would try to find the roles you would end up looking at an organizations management. 'The' Architect of an organization is its manager. He or she finally decides on how the business is organized, how it is run and what IT is implemented. But the field has become complex enough that a special function has appeared: the (Enterprise) Architect. The management has in fact outsourced the (rough) design of its solutions to a specialized function, whose task it is to handle all that complexity. Here at least is already one important point: Enterprise Architecture should be the responsibility of (organizational *management* of) the business, not of the IT provider. It is meant to help management to make fundamental decisions, not leave them to someone else with some requirements and then say "make it so".

Now, the Enterprise Architecture function has proliferated and also fragmented. There are now business architects, security architects, application architects, data architects, information architects, enterprise architects, infrastructure architects, domain architects, IT architects, solution architects, integration architects, the list seems endless. And to make matters worse: the same job name may mean quite something different depending on who and where you ask for the definition. What one company calls a business architect, the other company calls an enterprise architect or a lead architect and what one company calls an enterprise architect another may call information architect, etc..

So, I am going to lay out my definition and fragmentation of Enterprise Architecture. First, in line with the Enterprise Architecture modeling language ArchiMate, I divide Enterprise Architecture into the following layers:

- **Business & Information** Layer
- **Application & Data** Layer
- (Technical) **Infrastructure** Layer

Enterprise Architecture for me has nothing to do with the organizational unit (department, business unit, project) that the architect has as his *domain*, but everything with the fact that he or she is architect on all *layers*: business & information, application & data and infrastructure. *Enterprise Architecture is about the coherent design and modeling of all layers*. For me, a Project Architecture is also an Enterprise Architecture, because a project is also a domain and an 'enterprise' in itself.

I do recognize specialization of architects at a layer: a Business Architect is concerned with the business & information layer and an Infrastructure Architect with the infrastructure layer. Sometimes, these layers are called domains too, but I find that confusing. I reserve domains for recognizable divisions of the organization, such as departments, business functions or projects. So, for me, an Domain Architect is an Enterprise Architect within a certain domain.

If enterprise in 'Enterprise Architect' does not denote organizational level, how do we then call the chief enterprise



architect of the organization? My favorite job name for such a function is 'Lead (Enterprise) Architect' (and he or she should fulfill (amongst other things) a role comparable to that of the 'Lead Legal Counsel').

You can forget all of this, except for one thing: in the context of ArchiMate, 'Enterprise Architecture' says nothing about being in the top of the organization, but about the fact that an 'enterprise' is a coherent landscape that can be divided in business, application and infrastructure layers (or very roughly: people, software and hardware) and Enterprise Architecture is about *all* of them.

A second division, is often made in our field: a division between 'architects' and 'designers'. For me, there is no fundamental difference between the two, both are forms of design and the only difference is the level of detail they are concerned with. Leaving out details is not to be taken lightly. It is one of the most difficult aspects of Enterprise Architecture. In my view, an Enterprise Architect is concerned with all details, but sparingly goes into those details. Architecture is in part the art of leaving out *irrelevant* details. Leaving out details, though, often derails into religiously ignoring details. The key word is 'irrelevant': as the Chinese proverb says: people stumble over molehills, not over mountains. An architect consciously leaves out details that he or she has decided are irrelevant to the decisions to be made.

A long time ago I had to follow a basic course on safety at the multinational company I was at that time working for. Here I learned a very valuable lesson: working safely is not about *avoiding* risks, it is about *consciously taking acceptable* risks. There is no such thing as *not* taking risks. For me, the same applies for abstraction in design. *Abstraction* is not about *ignoring* detail, it is about *consciously leaving out* detail. And luckily, as we will see later, ArchiMate is equipped for that, as it has a mechanism that supports having coherent detailed and non-detailed views of the same model.

There is a third division one can make in Enterprise Architecture. Basically, an organization can use architecture in the following three settings:

The **Current-State** (or As-Is or IST) **architecture**. This is a descriptive model of how the current landscape of business and IT is. It can be used for reporting (e.g. to regulators) and analysis;

The **Future-State** (or To-Be or SOLL) **architecture**. This is a (rough) prescription on how the future landscape should be. It generally consists of both high level models and principle and guidelines for the more detailed work done in the line of moving towards the intended state;

**Change architectures**. These are the descriptions of what Change initiatives like projects will produce. A common form is a Project (Start) Architecture. Like the Future-State,

this is a combination of models and guidelines, but the detail should generally be comparable to the Current-State as the Change initiative actually results in a very specific change of that Current-State.

Enterprise Architecture is in the end about:

- Making good *choices* in the light of the strategic goals and positions of your enterprise;
- Making coherent *choices* across your enterprise;
- Making good *choices* in themselves (e.g. in sense of total cost of ownership, etc.)

In all of these, modeling your existing state and your choices in ArchiMate can be really helpful.

## 2.1 Where are the principles and guidelines?

You might wonder, with all this talk about modeling: where are my architecture principles and guidelines the architecture is about and that guide development for instance? I can say this about it here:

There are many definitions of what Enterprise Architecture is. The widely quoted ISO/IEC/IEEE 42010 standard, for instance, defines (system) architecture as

*fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution*

There are two aspects: the *design itself* (elements, relationships) and the *principles of design*. This book is about the first aspect. I know many Enterprise Architects who are of the opinion that it is all about the second aspect. I could not agree less and hence my approach to Enterprise Architecture is above all about the *actual* Business-IT landscape decisions that have to be made. Whether the use of principles and guidelines is a good way to come to such decisions is a question that is outside the scope of this book. It is the subject of another book I might try to finish in the foreseeable future...

## 2.2 Disclaimer

I have to warn you: in case you did not know: you almost can't learn anything from a book (the exception being do-books like Bobby Fisher's *Chess Lessons*). The only way to learn something is by *doing* it. Knowledge is about 'know how' not about 'know what'. It's funny that — while being convinced of this — I have written a *book* to teach ArchiMate. But anyway, *you won't really learn ArchiMate from this book*, you need to seriously *use* the language to really learn it. But if you do that, this book is meant to help you out and speed you up.

# 3. Gratitude

I am very grateful for the assistance of the following people:

- First and foremost: Jos Knoops, colleague at APG, who has been an invaluable sparring partner when discussing how we would be using ArchiMate, e.g. patterns;

- Leon Joosten, colleague at APG and my then-time boss, who supported me at APG when I wanted to introduce ArchiMate to the organization. Without that opportunity I would not have been able to gather the experience to be able to fill this book.

- Peter Spiers of Adobe Forums, who helped me get started with Adobe InDesign and helped me solve serious issues in my first attempts at the document and without that help, I would not have a tool to produce this properly.

## 4. License

This book is © Gerben Wierda. All photographs used are public domain. All diagrams have been created by Gerben Wierda. All product names mentioned are Registered Trade Marks of their respective owners.

I originally planned to sell this book for around US\$10 or €9 (money value 2012) in electronic form, including interactions like questions and such. However, I was unable to get that working technically so I fell back to a printed book. Getting it distributed in print, with probably the limited amount printed (at least initially ;- ) would make the book very expensive. So, instead, I am offering this book as an electronic download with the following license:

- If you want to use this book privately and non-commercially (e.g. learning ArchiMate by yourself, not as part of a business), you are hereby granted permission to make your own print or use your own electronic copy under 'shareware' fees: you are encouraged (free to, but not obliged to) donate an amount to me. The preferred amount is €8.99 (money value 2012) per copy
- If you want to use this book commercially (e.g. for internal training purposes or in the line of work for a professional organization), you are only allowed to print it or use the electronic copy after having received

- And of course, last but certainly not least: my family, who — while I was writing — have carried the burden of the fact that I was writing and thus had less time to do my chores and be a sunny, active part of family life.

permission by me. Permission will be granted based on a fee *per printed copy* or *per user of the electronic copy*, not exceeding €8.99 (money value 2012) per copy.

- If you want to print and distribute paper copies (e.g. for profit) you have to get permission from me for a fee per printed copy. The fee depends on your selling price and number of copies.
- On demand, I will print, bind and ship copies of the official print edition: ISBN 978-90-819840-1-0. These will cost ±€65 + transport + (potentially) taxes.

You can contact me at the following e-mail address:

[info@masteringarchimate.com](mailto:info@masteringarchimate.com)

You can donate via [this link](#).

Printing: You should print in high resolution, two-sided, in color and edge-to-edge.

Binding: a ring-shaped binder works well. It lets you let the book lie perfectly flat for easy reference.

Heerlen, The Netherlands, 15 October 2012

Gerben Wierda

## 5. Release Notes for Edition I

This document is the first edition of my book. Some things are not complete and there are still omissions. Most importantly:

- Hardly any attention has been paid yet to structuring passive objects (information & data structure)
- Maybe, more explicit attention should be paid to patterns regarding the application–business relation (application use patterns, next to application deployment patterns).
- ArchiMate's *Implementation and Migration Extension* is not described.
- ArchiMate's *Motivation Extension* is only summarily described. Its concepts are only used to create patterns for modeling Risk and Security.
- I did not get around yet to describing patterns for modeling Citrix-like infrastructure services.
- View numbering is slightly chaotic. This is the result of my tool (InDesign) which cannot handle a combi-

nation of floating and anchored/in-line figures well. I've tried to make sure that for references to views on other pages, the page number has been added to the reference.

- There are only a few questions and answers. In a future edition, I plan to have more.
- 3 November 2012: repaired broken link and license, no changes in text.
- 11 November 2012: created copies that conform to ISBN Guidelines, no changes in text.
- 3 December 2012: Fixes for some typos. Fixed error in texts regarding Location.
- 5 December 2012: Fixed errors in text regarding to Product and Derived Dynamic Relations.
- 5 January 2013: Fixed illegal relations in Views 150, 151 and 154.



This page intentionally not left blank

# Architectural Basics





# ArchiMate Basics

## I. Objects and Relations: 3x3x3

When modeling Enterprise Architecture, we need a language that knows about the concepts of Enterprise Architecture, and ArchiMate does a good job. To start, you need to know that ArchiMate is built from three types of *objects*:

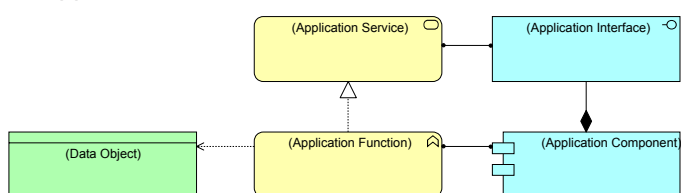
- objects that act (**active** objects)
- objects that represent the behavior of those 'objects that act' (**behavioral** objects)
- objects that cannot act and which are acted upon by that behavior (**passive** objects)

The three object types, connected by *relations*, can form sentences of sorts. A pickpocket (the application) steals (the application function) a wallet (the data). This is what makes ArchiMate into a grammar/language. Your model is a story of sorts. It tells the reader the basic structure of the story: **who** **acts** on **what**.

Let's start somewhere in the middle, at the application & data level of Enterprise Architecture.

### 1.1 Applications and Business

An application is modeled in ArchiMate as seen in View 1.



View 1. The Basic Application Pattern

This is possibly the first snippet of ArchiMate you have ever encountered, and it already has 5 object types and 4 relation types, so we are going to take our time describing it.

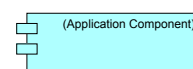
The two yellow and two blue objects in the image together make up the application. The green object is the data the application operates on. Think of it as the blue & yellow objects representing the word processing application and the green object the document being edited. The lower three objects represent the *internals* of the application and

the data. The two upper represent how the application is used by and is visible for a user.

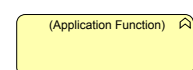
One of the most essential aspects of ArchiMate is that modeling the behavior is separated from modeling the structure. The blue objects in the figure represent the active structure ('who') and the yellow objects represent the behavioral aspects of the 'who' objects. But: they always exist together, they are two sides of the same coin.

It seems rather excessive that you need four objects to model one application. We will later see that you can simplify this, even to a single one of these objects, but for the moment it is very important to understand how the underlying structure is. Actually, the lack of addressing this in documents and courses I have seen, has been a major reason for writing this book. The understanding of the foundation is required to model *well* in ArchiMate.

Having said that, here is a little bit of explanation of the 5 object types in the image:

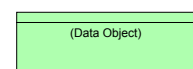


This is the **Application Component**. It stands for the 'actor' that an application in your Enterprise Architecture landscape actually is. It is one side of the coin of which Application Function (its behavior) is the other side.



This is the **Application Function**. It stands for the behavior of the Application Component, how the application can act. It is one side of the coin of which Application Component is the other side.

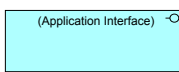
Application Component and Application Function are in fact inseparable. You cannot have an actor that does not act unless the actor is dead, and in that case it should not appear in your architecture. Without magic, you cannot have an act without an actor. Again: later we will see how to leave things out of our views and models, but for now we stick to the details.

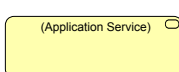


This is the **Data Object**. It is what the Application Function acts on. The Application Function might create, read, write, update, delete the Data Object. Conceivably, the Data Object might not be needed, you can imagine behavior that does not access a passive object. You can

also imagine that you do not model Data Objects that only exist *inside* your application. That variable in the application code is not modeled. But as soon as the Data Object is visible to other parts of your landscape, or when it is persistent, it should be there. Generally, that means we only model (semi-)persistent Data Objects. Note, this is not the file or the database itself, those are represented on a lower level: the infrastructure level. We are still one abstraction level up. To illustrate the difference: an RTF file can both be a MS Word data object or an Apple TextEdit.app data object, depending on which application is used to access it.

Two objects in the image have not been explained yet. They have to do with how the application is used/seen (by the business or by other apps):

 This is the **Application Interface**. It stands for the route via which the application offers itself to the business or to other apps. Note: both separate concepts (used by people and used by other apps) are supported by this one object. One example would be a Graphical User Interface (GUI), but it can as well be an Application Programming Interface (API), a Web Service or one of the many other ways an application offers itself to other 'actors' in your landscape. Given that difference in use (and thus, as Uncle Ludwig would say, meaning), it is unlikely that the same interface will be used by both a human or another application. But it can be, e.g. in the case of a Command Line Interface (CLI) used by a scheduler system. Application Interface is a 'handle' of the 'actor' that is the Application Component. It is one side of a coin of which Application Service is another side.

 This is the **Application Service**. It stands for the 'visible' behavior of the Application, how the Application Interface can act *for a user*. It is one side of the coin of which Application Interface is the other side. Here again, the service may be 'technical' in that it is offered by one application to other applications or it may be part of your Business-IT integration: services offered to business processes (behavior of humans). The same type of object is used for both.

Now, apart from the objects, there are relations between the objects. There are four in the initial image:

..... This is the **Access** relation. The access relation always depicts a behavioral object accessing a passive object. Here it depicts the behavior of the application (its function) accessing a passive data object (e.g. something that in the end resides in a file or a database). The arrowhead is optional and it may depict read access or write access (e.g. two for read/write).

◆ This is the **Composition** relation. It means that the object at the end with the diamond is the *parent* of the object on the other end and that the child *cannot exist independently* from the parent. The relation depicts the composition of larger wholes out of smaller parts, but it does not mean the set of children modeled must be necessarily complete in your model: there may be parts that are not modeled. This relation could

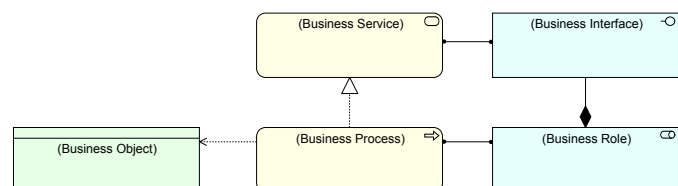
also for instance be used to show that an Application Component has various subcomponents.

..... This is the **Realization** relation. This has two types of uses in ArchiMate. Here it means that the object at the end without the arrowhead is the object that 'creates' the object at the end with an arrowhead: the app's internal functionality realizes a service, which is the externally usable functionality of the application.

— This is the **Assignment** relation. This also has more than one meaning in ArchiMate. Here, it means that one side (the active object) *performs* the behavior that is the behavioral object on the other side.

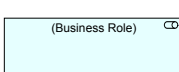
Important and possibly initially confusing aspects of ArchiMate are thus that — while we have multiple objects (structural and behavioral) representing what is in the mind of many a single thing (an application or an application interface) — we also have single objects (and as we will later see, relations) that can be used with multiple meanings. When you get the hang of it, it all becomes pretty natural just like it is with natural language, but if you are looking for a strictly disjunct (made of independent concepts) approach to a modeling language (e.g. like a programming language or like UML), it might be confusing in the beginning.

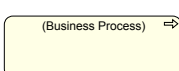
Having modeled an application, we can turn to modeling the way this application is used by the business. And before that, we need to look at the way the business & information level of Enterprise Architecture is modeled in ArchiMate. Luckily, it looks a lot like what happens on the application level so it is easy to understand now and it can be seen in View 2.



**View 2. Basic Business Process Pattern**

I have left out the actual human actor, the one that fulfills the Business Role, for now to stress the equality between this pattern and the one about the application in View 1 on page 15. It looks quite the same and that is not a coincidence. The relations are the same as with the application image above. The new object types are:

 This is the **Business Role**. The Business Role is an 'actor' in ArchiMate, but it is slightly more complicated than that, because it is an abstract sort of actor based on being responsible for certain behavior. The real actors are people and departments and such. ArchiMate has an object type for those as well, but we leave that for later. Business Roles can perform Business Processes.

 This is the **Business Process**. It stands for a set of causally related activities that together realize services or create objects. Roles can be assigned to a process, they perform the process, just as the Application Component performs the Application Function. Just as on the

application layer, a Business Process cannot exist without a Business Role (which does not mean you must model both, I am talking about the reality you are modeling), they are two sides of the same coin.

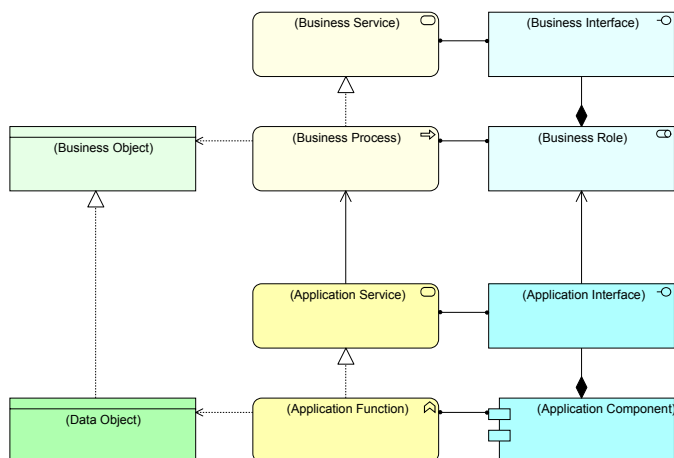
**(Business Object)** This is the **Business Object**: the abstract object that is created or used by a Business Process. Think of objects like 'a payment' or 'a bank account' or 'a bill'. Though it is named *Business Object*, it is more like a *Concept*.

Again, just as with the application, these objects can make sentences of sorts. The proverbial 'second hand car sales' role performs the 'sell second hand car' process, which creates a 'bill'. Criminal, crime and — in this case, possibly — proof of crime.

**(Business Interface)** This is the **Business Interface**: the way the role interacts with others. You can think of it as a 'channel'. E.g. phone, mail, meeting, etc.. The interface is the visible manifestation of a role.

**(Business Service)** And this is the **Business Service**: more or less what it is always about in an organization. This is the reason for the existence of the process. This is the service it offers to (and thus can be used by) others (either inside or outside the company). A company might offer many services. E.g. a bank offers at the most abstract level a 'savings' service or a 'loan' service.

Having set up a couple of basic objects and relations, we can now fill in the missing links. Because on one hand we have the business that offers services to others, on the other hand we have IT that offers supporting services to the business. Together they look like View 3.



**View 3. Basic Application is used by Basic Business Pattern**

The application level is connected to the business level by three relations. On the left we see the already familiar Realization relation (.....>). Here it means that the Data Object realizes the Business Object. In a concrete example: the 'bank account' Business Object may be data (a Data Object) in an accounting application, it is the same item's representation in a different architectural layer.

In the middle and on the right we see a new relation:

← This is the **Used-By** relation. It means that the object at the end without the arrowhead is used by the object at the end with the arrow head. The Application Service for instance is Used-By the Business Process. The Application Interface (e.g. the Graphical User Interface) is Used-By the Business Role. Note especially that the definition is passive: it is not 'uses' but 'is used by'. The reason for that is that the direction of relations is important in ArchiMate, something that will be explained later on.

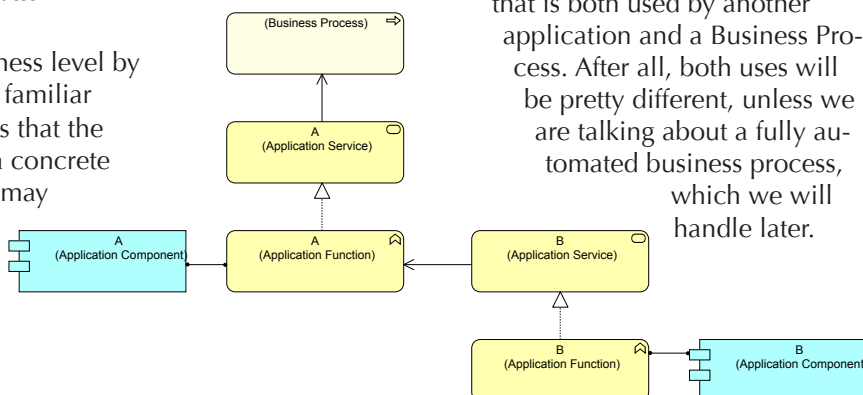
The fact that an Application Interface can be used by a Business Role is the 'other side of the coin' of the same Used-By relation between Application Service and Business Process. They are twins. Both illustrate the 'service oriented' way that ArchiMate relates one layer to the next as far as actors and their behavior goes.

With what has been explained so far, you can already do much of the modeling you need in terms of Current State or Project Enterprise Architecture. There are two more things that need to be explained before the basic setup is complete: applications using other applications, and business processes/roles using other business processes/roles. Here again, what happens at business level and application level is identical, so we are going to illustrate one.

## 1.2 The double use of Used-By

So far, our example has only shown Used-By as a relation *between* levels in your architecture. But the same relation type can also be used *within* a level. The business may use an application, but an application can also be used by another application.

In View 4 you see the same Used-By relation (←) twice. Once between the 'A' Application Service and the Business Process that uses the 'A' application. And once between the 'B' Application Service and the 'A' Application Function. This means that the 'A' application makes use of the 'B' application. Though the relation is Used-By in both cases, it has quite a different role to play. Often, the definition of an Application Service that is used by the Business is pretty business-like in its description, something you discuss with a senior user or a process owner. But the relation between applications is more of a technical nature and you discuss it with application architects. The difference generally shows itself clearly in the types of names and descriptions of the service. It is hard to truly imagine an Application Service that is both used by another application and a Business Process. After all, both uses will be pretty different, unless we are talking about a fully automated business process, which we will handle later.

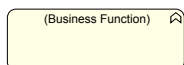


**View 4. An Application Using Another Application**



## 1.3 Business Function

So far, we have consequently modeled business behavior as a Business Process. But ArchiMate actually has two main work horses for business behavior: Business Process and Business Function. The differences are subtle and in most situations you can use either. In 10 “Business Function and Business Process” on page 71 we will look into the difference in more depth and see that there is a nice way of combining them both when modeling your business layer. **Business Function** looks like this:

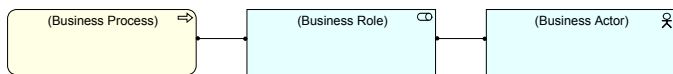


For now, as we are doing the superficial introduction, it is best to use the following guidelines:

- You use a Business Process if you are thinking of a causally related set of behaviors (‘activities’) that in the end *produce* something, normally a Business Service. Business Process is an outside-in view of business behavior, based on what the behavior produces. You assign a single Business Role to a Business Process or Business Function, but that leaves you free to have sub-roles assigned to sub-processes or sub-functions. If disjunct roles do something together, you should (officially) use a Business Interaction in ArchiMate (see Section 2.1 “Collaborations and Interactions” on page 23) but you often have quite some freedom to choose a different ‘not-proper’ pattern (we’ll get into this later).
- You use a Business Function if you are thinking of a grouping of related behavior based on — for instance — same tools, same skills or same role that performs it. In fact, Business Function is best seen as an inside-out view of the business behavior.

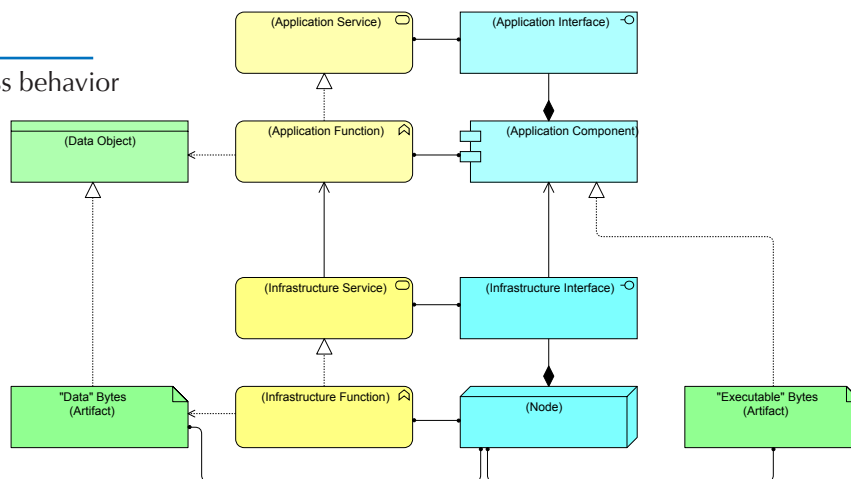
## 1.4 Business Actor

Earlier we encountered the Business Role. The Business Role is an abstract sort of actor. But in a business architecture there are of course *real* actors: people, departments, or even business units or companies or maybe regulators. ArchiMate has an object for that: the Business Actor. In a context seen in View 5.



View 5. Business Actor in Context

On the left we see the already familiar Business Process to which a Business Role is Assigned. The Business Role *performs* the Business Process. On the right we see the new object type Business Actor, which is Assigned-To the Business Role. This must be read as the Business Actor *fulfills* the Business Role. ArchiMate was designed to be economical with relation types, so they re-used the relation for a slightly different meaning.

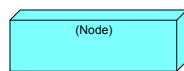


View 43. Basic Application uses Basic Infrastructure

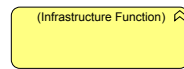
## 1.5 Adding Technical Infrastructure to the Mix

Just as the Business Process needs the Application Service to be able to be performed, the Application Function needs infrastructure to run. If we add the infrastructure to the application layer, we get View 43.

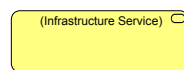
There are no new types of relations here, but there are new object types:



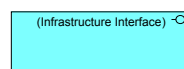
This is the **Node**. This is a slightly complicated concept in ArchiMate and more details follow later, but for now, think of it as the hardware and its system software, where files are stored or applications can run. We’ll see the details later.



This is the **Infrastructure Function**. Just as with the Application Function and the Application Component, the Infrastructure Function stands for the behavior of the Node. They are two sides of the same coin. Note: Infrastructure Function is new since ArchiMate 2.0. If you use older tooling or you encounter older views, it might not be there. Instead, in ArchiMate 1.0, the Node directly Realizes the Infrastructure Service. ArchiMate 2.0 did the right thing and made sure all layers have the same basic structure.



This is the **Infrastructure Service**, the visible behavior of the Node. In many ways, this is what the infrastructure is all about, this is what it can do for the applications, its reason of existence. This is what the applications need to function. Typical examples of infrastructure services are for instance a ‘file share’ or ‘application execution’ or a ‘database service’. The latter might raise your eyebrows now, because you might wonder why that is an infrastructure service and not for instance something at the application level. We’ll get back to that in Section 5.6 “Why two types of software?” on page 35, but for now, it is enough to say that the Node comprises both the hardware and the system software. A database system is generally modeled as system software and the database as an Artifact (see below).

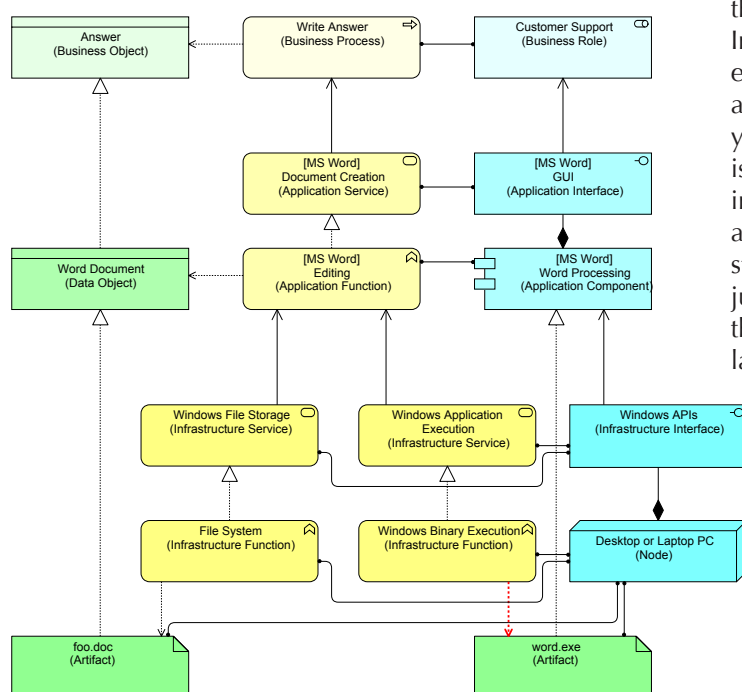


This is the **Infrastructure Interface**. This is not that easy to explain. For Application Interface, one can easily dream up an easy

example: the (graphical) user interface. But for infrastructure interface, ArchiMate is not very clear, as it says it might be best thought of as a kind of contract that the 'user' (the application) has to fulfill. An example would be a protocol, like the SMB or NFS protocol for file sharing and the TCP/IP or UDP/IP ports where the service is offered. Unless you are a dedicated Infrastructure Architect, you can generally do without this one. I am just mentioning it here to be complete and leaving it out here would introduce the concept of 'pattern' before it is wise to do so.

**(Artifact)** The last new object (for now) is the **Artifact**. This one is pretty simple. The best example is a file. Another often-used example is the actual database where your application's data resides. Another example is the actual 'executable' (file) also known as 'the binary' that is what your application is when you look at the 'byte' level. Your application. The 'data bytes' Artifact in the model above is the one that realizes the Data Object that is accessed by the Application Function. The 'executable byte' Artifact are the bytes (a file, or a set of files often called a 'software distribution') that the system can read and interpret as a program. On the infrastructure level, 'a byte is a byte' in the sense that both passive (Data) objects and active (Application) objects are in the end nothing but a collection of bytes. Deep down, we get to the basic level of zeros and ones as we should.

The relations are mostly pretty straightforward. The Assignment relations (●—●) between Node and Artifacts stand for the fact that the Artifacts *resides* on the Node. In other words, it depicts where in your infrastructure you can find a file. Also pretty simple are the Used-By (←) relations between Infrastructure Service and Application Function and between Infrastructure Interface and Application Component. If an Application Function needs access to a file that resides on a file system, the file system is an Infrastructure Service that is Used-By the Application Function. And its mirror is the Used-By



**View 6.** Write Answer Process, supported by MS Word and a Standalone PC

relation from Infrastructure Interface to Application Component. This mirroring is depicted by the Assignment relation (●—●) between Infrastructure Interface and Infrastructure Service, exactly as happens in the levels above between the 'interface' and the 'service'.

And lastly, there are the Realization relations (.....▷) between an Artifact and both Data Object and between Artifact and Application Component. This one is also pretty simple and easiest to explain by example. Suppose your application is Microsoft Word, then the file you are editing could be called "foo.doc". And if the application is MS Word, the Artifact realizing the Application Component could be "word.exe".

I have a detailed example model for you in View 6 to see everything in a single context. The model shows someone writing a letter to a customer that contains an answer, supposedly to a question the customer has asked. Two infrastructural services are needed for this to work. The application should run and the document must be stored. In this example, everything happens on a standalone PC.

Now I can imagine, if you are an 'enterprise architect' you think all this detail is irrelevant and the language looks like a language for detailed design. Don't worry, using ArchiMate does not mean you absolutely must model all these details, the language can handle both: a roughly sketched Business Operating Model down to nitty-gritty details you have to confront when you are in a project. I am just using an example everybody knows, to illustrate how it works in ArchiMate. And even in View 6, I have left some things out, and I have combined the Infrastructure Interfaces in one object. I could also have combined the Infrastructure Services in one object, this is a question I will address later when I am discussing 'modeling patterns'.

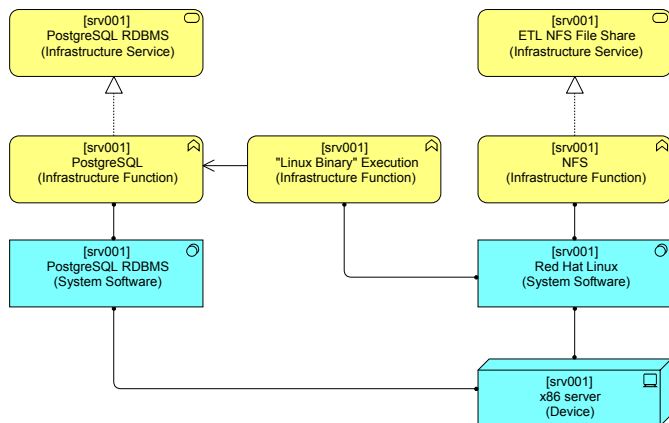
One more remark, before we go on with the rest of the language. As you might have noticed, the initial example with infrastructure in View 43 had no Access relation between the Infrastructure Function and the 'executable' Artifact. But the 'write answer' example of View 6 does have that kind of a relation (shown in red). Here an explicit Infrastructure Function for application execution was modeled and that function needs access to the 'executable' artifact. So which one is correct? The answer is: whatever your message is and what you want to show. ArchiMate is a language, but it is your choice what you want to say in that language and how you say it. You have considerable freedom and you will certainly develop your own style. You can certainly model incorrectly in ArchiMate, just as you can write false statements in any language. So there are wrong models. But the syntax of the ArchiMate language does not by definition lead to correct statements, just as with other languages. Later, we will address choosing your style and patterns, the latter being like 'common phrases in the language'.

## 1.6 System Software and Devices

Our first use of infrastructure objects was pretty limited. We added the Infrastructure Function, the Infrastructure Service, the Node, the Infrastructure Interface and the Artifact. ArchiMate adds two useful active structure concepts to model the actual infrastructure layer of your architecture: Device and System Soft-

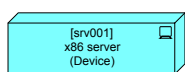
ware. The best way to explain them is by giving an example of how they can be used.

In View 7 on page 20 we see a model of a database server in our landscape. The name of the server is 'srv001' and that could be the name it carries in a CMDB for instance.

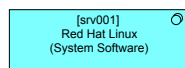


View 7. Device and System Software

The new object types are:



This is the **Device**, the actual hardware, the 'iron' of our infrastructure. In this example, two objects of the type System Software have been installed on it (Assigned-To it). The little logo of the object is an image of a keyboard/screen.



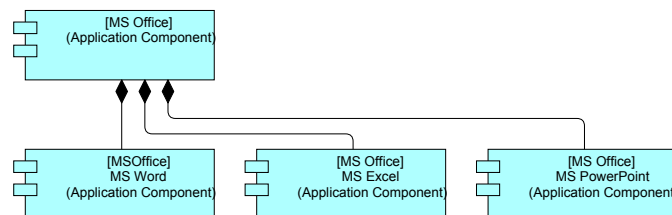
This is the **System Software** object. It stands for software that we consider to be part of our infrastructure layer and not our application layer. Most common uses are operating systems or database systems. In our example both are available: the Red Hat Linux operating system and the PostgreSQL database system. Modeled too is that the PostgreSQL software uses the Red Hat software. (If we have our existing landscape modeled in such detail, we could by analysis find all PostgreSQL databases that run on Red Hat, handy if we are planning an update and there is something about the combination).

Both are a type of Node. A Node like in View 43 on page 18 can be either System Software or Device, or a collection of both (see below).

## 1.7 Composition and Aggregation

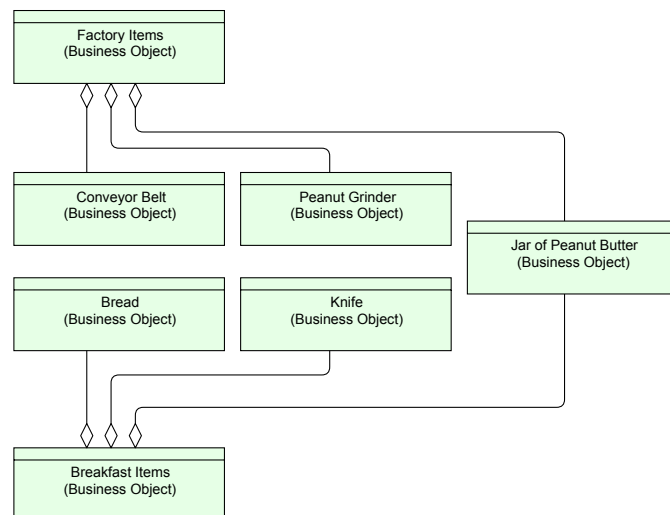
We already met the Composition relation (◆) earlier. The composition represents a whole-part relation. It is best to look at the ArchiMate version of this common relation as the relation between a tree (a real one from your garden, not the computational concept) and its branches. A branch is branch of a single tree and cannot be a branch of multiple trees. If the tree is destroyed, all of its branches are destroyed as well.

Generally, you may always create a Composition relation between two objects of the same type (or 'super-type', explained later). View 8 contains an example.



View 8. Composition Example

The Aggregation relation (◇) is a variation on the theme. It looks like View 9.

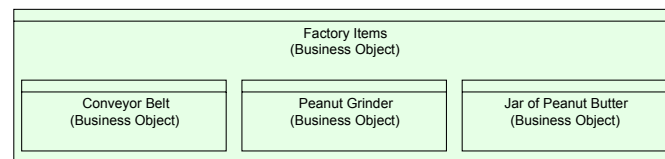


View 9. Aggregation Example

It is best to look at the Aggregation relation as a kind of grouping (note: in Section 2.8 "The Grouping Relation" on page 26 we will see an official 'grouping' relation). The 'parent' (on the end with the diamond) represent a 'collection' of the children. But other than with Composition, the children may be a member of multiple Aggregations as you can also see in View 9: The 'Jar of Peanut Butter' is both part of the 'Factory Items' of the peanut butter factory and the 'Breakfast Items' of a consumer's home. It's a like the number 4 being both part of the collection of squares of integers and the collection of even numbers.

## 1.8 Nesting

There are three relation types that may be drawn by nesting an object inside another object: Composition, Aggregation and Assignment. Note: tooling often allows more than the language does. Especially if you use tooling that is nothing more than a good model-drawing application like Visio for Windows or OmniGraffle for Mac OS. Anyway, let's take the 'Factory Items' from View 9 as an example. Nested, it looks like View 10.

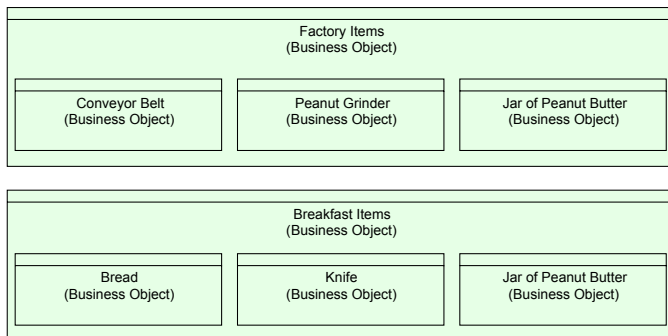


View 10. Nested Aggregation

That looks a lot cleaner and that is why many modelers like it. But there is already a disadvantage you can see: you cannot see anymore what the relation is between parts and whole: Composition? Aggregation?



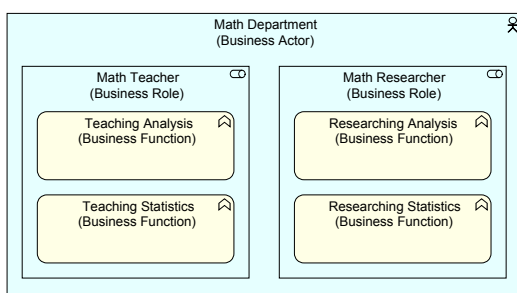
It gets worse, when you want to model both Aggregations from View 9 in one view as in View 11:



**View 11.** Two Nested Aggregations with Shared Object

Not only are you unable to see the actual relations, it has now become necessary to include the 'Jar of Peanut Butter' object twice. And though the name is the same, there is nothing that will tell you if 'under water' it is the same object. You can have two different objects with the same name in ArchiMate, after all, the label has no meaning inside the language (as the language' is in fact a *grammar*) even if it has a meaning in the world of an architect. Besides, even if your modeling guidelines force different names for different objects, think of a very large view with the same object twice. Are you going to spot that the same object occurs twice? Probably not. So are you going to see the dependencies in full? Again: probably not.

View 12 contains an Assignment example of Nesting, two levels deep:

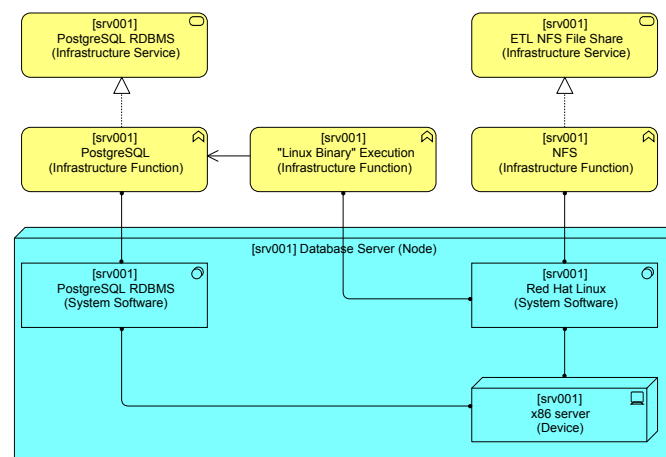


**View 12.** Assignment Nesting, Two Levels Deep

Summarizing: Nesting makes for nice views on your model, views that are easy on the eye. But don't think about them too lightly, because constructing your model this way comes with risks. And even bigger risks than you think, because some tools will let you create nestings without actually creating a relation between the nested objects (which is in conflict with ArchiMate), or they may create a default relation which was not the relation you were thinking of.

## 1.9 Using a Node to encapsulate infrastructure

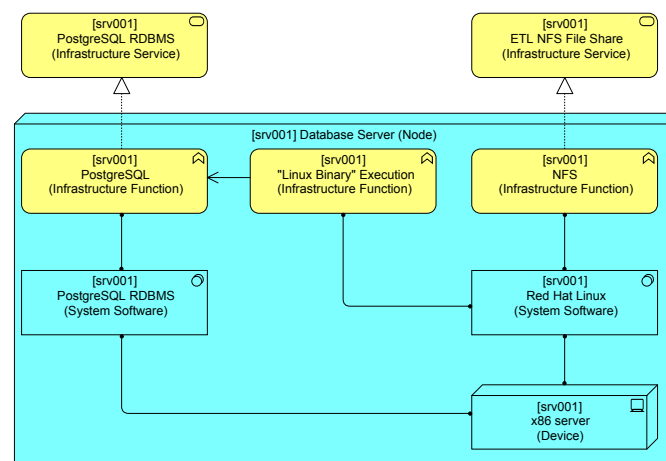
Now that we have seen Composition and Nesting, we can look at View 7 on page 20 that introduced System Software and Device and show in View 13 how it might be modeled.



**View 13.** Device and System Software Nested in a Node

In this example, the System Software and Device objects are children (Composition) of an abstract Node object (the Composition relations are not explicitly shown here, but instead modeled as Nesting). This is kind of a nice grouping of an infrastructure element as the composition's children have no independent existence or use. We can also go further: in View 14, the Infrastructure Functions too have been Nested in the Node. This is possible, because we can also Assign the functions to the overall Node and we may Nest an Assignment relation.

In this example, the Node realizes two Infrastructure Services: the PostgreSQL software realizes a RDBMS service that is available to Application Functions and the operating system also realizes a network file system shared directory. Basically, all structure below the Infrastructure Services is modeled inside the Node. Later, we will go one step further with this example when we discuss patterns.



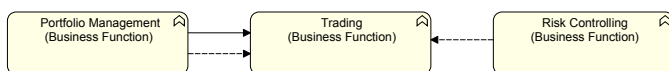
**View 14.** Device, System Software and Infrastructure Function Nested in a Node

In Section 7.2 "A Basic TI Pattern: A Database" on page 44, we will see that it might be handy to restrict ourselves here, to make analysis of the model easier. You have the choice of course if you want to model these details

(and you can go as far and deep as you like). It all depends on the use you want to make of your model (in other words: uncle Ludwig was right). Here, it's only shown to explain what the object types are and how they relate to each other.

## 1.10 Events, Triggers and Flows

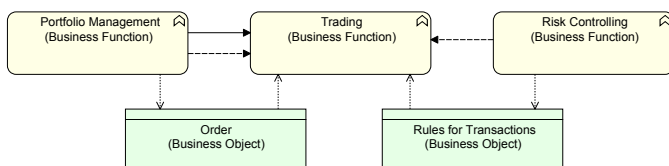
So far, we have handled the 'structure' of your architecture and all relations so far were so-called 'structural relations'. ArchiMate is not big on the dynamics of an architecture, but it does have two 'dynamic relations': Trigger and Flow, and one operator to combine these into a dynamic structure (see Section 2.9 "The Junction Relation" on page 27).



View 15. Trigger and Flow

In View 15 we find **Trigger** (—→) and **Flow** (---→) relations in a business layer example. We see three Business Functions here from the asset management world: 'Portfolio Management' is taking investment decisions, these result into orders for the 'Trading' function. So, 'Trading' starts trading when it receives an order from 'Portfolio Management'. Triggering means there is a causal relation between the two functions. The flow between 'Portfolio Management' and 'Trading' says information flows from one to the other. In this case it is the 'Order'.

But the Trading function also regularly receives a list of allowed counterparts, countries and currencies from the 'Risk Controlling' function. This information Flows from one to the other, but it does not Trigger a trade. The Flow relation between the Business Functions can also be modeled as Business Objects written (Accessed) by one function and read (Accessed) by the other. If we add those, it looks like View 16.

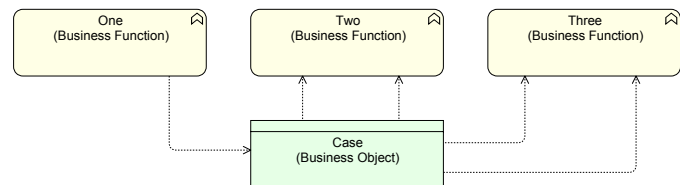


View 16. Trigger, Flow and Access to Objects

So, how useful is the Flow relation if you can also use the Business Objects and the Access relation? Well, it has a few advantages:

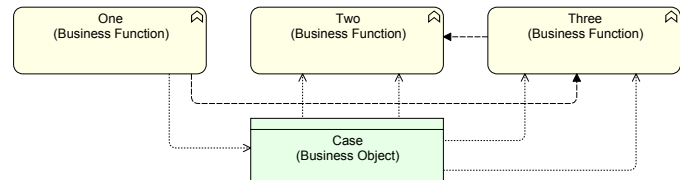
- You can make simpler view by leaving the Business Objects out and for instance label the Flow relations. The problem, though, is that such a label generally cannot be used to analyze what goes on your landscape, I'll say it already here: watch out for relying too much on labels and properties of objects, they often live outside the 'analyzable' structure of your model.
- But most importantly, your dependencies can become clearer with a Flow. Take for instance the example of a 'Case' flowing through your business from Business Function to Business Function. Having read/write relations from these Business Functions to that single

Business Object tells you nothing about how information flows. Take the example in View 17:



View 17. Business Functions sharing Access to a Business Object, without Flows

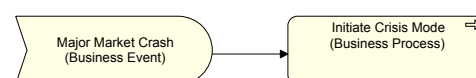
This does not tell you how the 'Case' Flows through your organization. What do you think? Look at View 18 and it becomes clear what the flow of information is.



View 18. Business Functions sharing Access to a Business Object, with Flows

Without the Flow relations, would you have known? Could you have drawn the wrong conclusion? Certainly. Is that a bad thing? After all they all depend on that Business Object. Well, take this example: without the Flow relation, you might think that the roles behind function One, Two and Three may have to agree concurrently on all the contents of the Case Business Object. But in reality you might only need to setup talks between One and Two on the one hand and Three and Two on the other and depending on the issues at hand, that might be simpler.

ArchiMate also has a special object that depicts a Business Event. A **Business Event** is 'something that happens'. Events can trigger a Business Process or a Business Function and they can be raised by a Business Process or Business Function, which is also depicted with a Trigger relation. Business Events are normally used for standalone 'things that happen'. An example can be seen in View 19.



View 19. Business Event Triggers Business Process

## 1.11 The (almost) complete Picture

We have described 14 of ArchiMate's 31 object types, so in that sense we are only half way. But with the objects and relations of this section you can probably do 95% of an architect's work, if that work is focused on normal business-IT integration and modeling things like Project Architectures.

The title of this section is "Objects and Relations: 3x3x3". It has this title because ArchiMate:

- divides Enterprise Architecture in a Business & Information layer, an Application & Data layer and a Technical Infrastructure Layer. These are the rows in ArchiMate's meta-model and it is quite standard in the Enterprise Architecture world.
- divides architecture in any layer 'strictly' into Active Structure, Behavior and Passive Structure. In a sen-

tence: “Who/what does what to what?”. The clear separation of actors and their behavior is not common in Enterprise Architecture and it is a main foundational aspect of ArchiMate.

- has three kinds of relations: Structural Relations, Dynamic Relations and ‘Other’ Relations (next section). The reason for this division becomes clear in the section on Derived Relations.

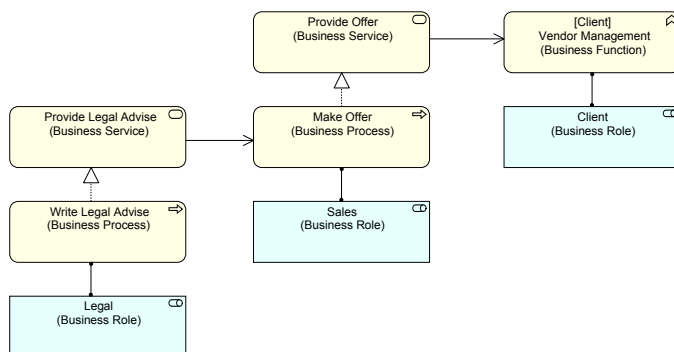
## 2. Other Objects and Relations

With the objects and relations of the previous section you can probably do most of an architects work, if that work is focused on Current State and Project Architectures. Here is the rest:

### 2.1 Collaborations and Interactions

Suppose you have two Business Roles in your organization that need to work together to get some specific work done. For instance, the ‘sales’ and the ‘legal’ Business Roles need to work together to create the offer for the prospective client. Within ArchiMate there are generally two ways of modeling this.

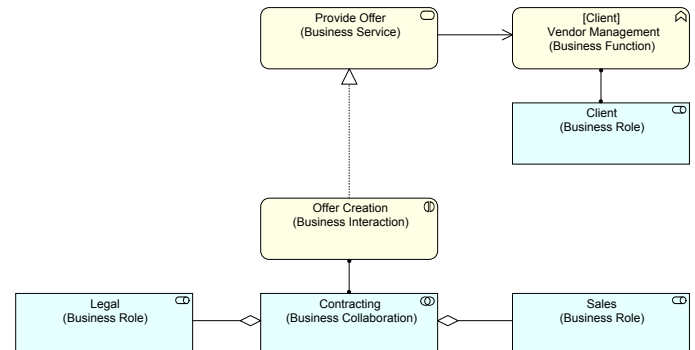
Using the objects and relations we already described, you can put one of them in the lead and let the second provide a service. In our example: ‘sales’ produces the offer, but it uses the (internal) services realized by (the processes of) ‘legal’ in its processes:



**View 20. Collaboration: Sales Uses Legal on Order Creation**

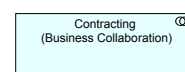
In this setup, it is clear who is in charge. ‘Sales’ decides to send the offer out, not legal. ‘Legal’ must provide something (and for instance withholding approval means ‘sales’ cannot proceed), but the service to the client is realized by the process of ‘Sales’.

ArchiMate offers a second way to model such collaborations. You can create a Business Collaboration that is made up of the Business Roles that are part of that collaboration. Then, you can assign a Business Interaction to that collaboration, the interaction being the behavior of that collaboration. It looks like this:

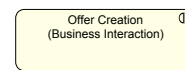


**View 21. Collaboration: Sales and Legal Collaborate on Order Creation**

The whole ‘offer creation’ ‘process’ is now modeled as not one of the parties ‘owning’ it, but both. There are a few new objects:



This is the **Business Collaboration**. It is a special kind of Business Role that Aggregates multiple other Business Roles (or Business Collaborations of course, as these are also types of roles, see Section 2.3 “The Specialization Relation” on page 24). It is the concept that defines multiple roles forming a (specific) single ‘collective’ role to do something together.



This is the **Business Interaction**. It is the behavior of that Business Collaboration, the activities they do together. Note that, while the Business Collaboration Aggregates two or more Business Roles, the Business Interaction does *not* Aggregate Business Functions or Business Processes. What is not clear in ArchiMate is if we want to see a Business Interaction as functional or process-oriented. Given that it generally produces a result, it is probably best to see this as process-oriented.

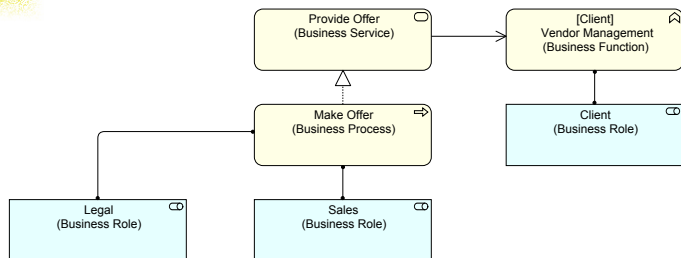
One consequence of using a Collaboration is that it is not clear who is in charge. This might be more acceptable to the organization in terms of sensitivities, but sometimes it is just true: nobody is really in charge. In Section 10.2 “Business Function or Business Process?” on page 72, I will present a ‘modeling pattern’ of the business that uses Collaboration to show the loosely coupled nature of some of the organization’s ‘end-to-end’ processes.

Both methods (use a service and interaction) are correct, it is a matter of style what you want to use and both approaches have some consequences. If you think about it, you could even model the sales-client interaction as a collaboration, after all the transaction requires decisions of both sides. For this introduction, it suffices that you have seen this.

Actually, there is a quick and dirty way to do this too. ArchiMate says that a Business Process is assigned to a single role, but it does not force this in the formal definition.



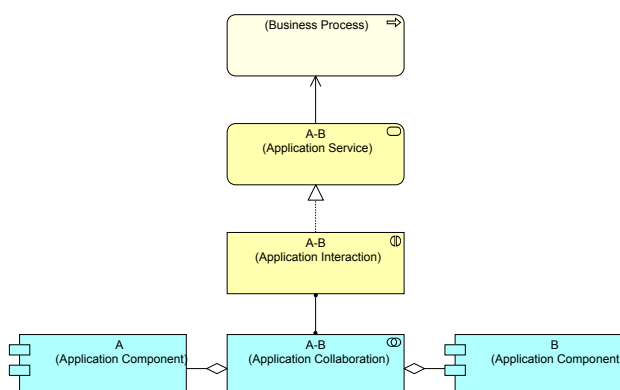
Nothing stops you from assigning multiple roles to a single process. So, we can model it quick-and-dirty as follows:



View 22. Collaboration: Quick and Dirty Method of modeling.

This way, we also show that the ‘Make Offer’ process is assigned to two roles, in fact forming a de facto collaboration. This is, however, not proper ArchiMate. Besides, why is this a collaboration? Maybe *either* of the two performs it (which is where I would use this pattern for, if at all).

At the application layer, something likewise exists:



View 23. Application Collaboration

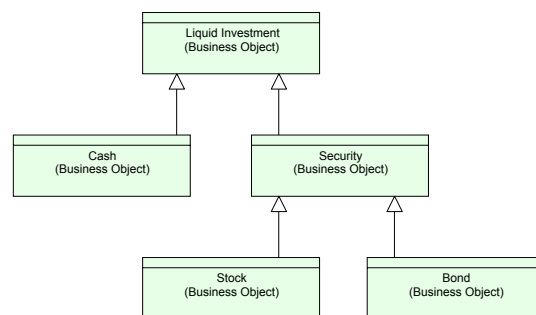
The **Application Collaboration** stands for the collaboration of two Application Components. The behavior of that collaboration is the **Application Interaction** and such a behavior may realize an Application Service. Here too, it is not clear who is in charge, but an architect may like it because it is ‘fairer’ to the importance of both. Also, it gives the architect a clear central place to document how applications interact. In the Used-By model, the description gets divided over multiple objects. This is a matter of style, and we’ll get back to that in Section 8.2 “Application Collaboration is an Anthropomorphism” on page 63.

## 2.2 The Association Relation

The weakest relation, the ‘catch-all’ relation of ArchiMate is the **Association** relation which is depicted as a simple line (—). It has a couple of formal roles in ArchiMate, but it also has the role of ‘relation of last resort’. If you want to model the relation between two objects, if you know they are related somehow but you cannot model how, you can use Association. It is, therefore, often a sign of lack of knowledge or effort, so it is a matter of style if you want to use it (we’ll talk about style later). For now, know that it exists and below we will see some official uses.

## 2.3 The Specialization Relation

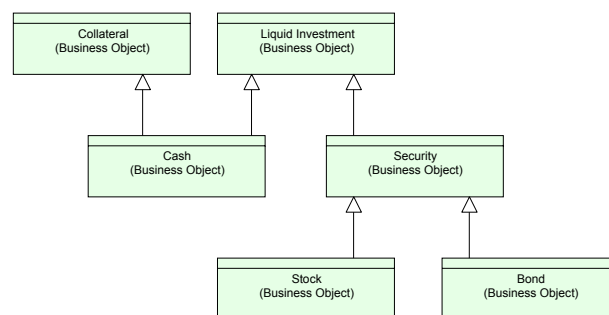
All relations we have seen so far are relation between actual elements in your landscape. The **Specialization** relation (◁) is different. For a programmer, I can explain it easily: all other ArchiMate relations are object-level relations, this one is a class-level relation. The Specialization relation says that an object is a kind of another object. For instance, a ‘car insurance’ and a ‘travel insurance’ are both a kind of ‘insurance’. Or a ‘stock’ and a ‘bond’ are both a kind of ‘liquid investment’. It may look like this:



View 24. Specialization Example: Investment Types

Here, we say that ‘Cash’ is a kind of ‘Liquid Investment’ (a liquid investment is an investment that you can easily trade) and another kind is ‘Security’ which again can be specialized into ‘Stock’ (a deed to a partial ownership of a company) and ‘Bond’ (a loan to an organization or country). If an object is a specialization of another object, whatever relations are true for the ‘parent’ (the object at the arrowhead) must be true for the child. If a Business Process handles ‘Liquid Investment’ (e.g. a reporting process), it must be able to handle both ‘Cash’ and ‘Security’. On the other hand, if a business process handles ‘Security’, it must be able to handle both ‘Stock’ and ‘Bond’ but it does not need to be able to handle ‘Cash’.

In Object-Oriented Design, the specialization is sometimes called the ‘Is-A’ relation. Its counterpart is the ‘Has-A’ relation, which in ArchiMate is either Composition (◆) or Aggregation (◊). Generally, what you can do with an ‘Is-A’ Sub-classing (which is what you do with the Specialization relation), can have complex consequences. Take for instance ‘Cash’ from the current example. Suppose our company says we may only use cash as collateral for securities we lend and not (other) securities. We would have a new Business Object in our landscape called ‘Collateral’ and ‘Cash’ could be a kind of ‘Collateral’.



View 25. Specialization Example: Multiple Inheritance

What we have here is called in OO-terms ‘multiple inheritance’: Cash is both a kind of collateral and a kind of security. Most OO languages do not support true multiple inheritance (C++ does, more modern Objective-C, Java and C# do not) and for a reason: it tends to become messy because the ‘parents’ may have conflicting rules of behavior for the ‘child’ to inherit (a bit like parenting in real life, true, but maybe not the best paradigm for software engineering or business object modeling).

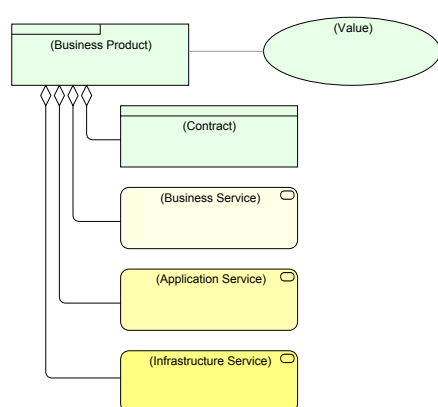
What is interesting to know is that Specialization also plays a role inside the ArchiMate meta-model. For instance, Device and System Software are both Specializations of Node, a Business Collaboration is a Specialization of Business Role and an Application Collaboration is a Specialization of Application Component. What this means above all is that as you can have Compositions and Aggregations between objects of the same type, and, for instance, since Device is a subtype of Node, you can have a Device as Composite part of a Node (as in Section 1.9 “Using a Node to encapsulate infrastructure” on page 21.

There are also hidden specializations in ArchiMate. In the specification, Business Process, Business Function and Business Interaction are all specializations of a common, invisible ‘business behavior’ concept. As a result, all these ‘business behavior’ objects may be Composites and Aggregates of each other.

In Section 7.21 on page 60, I’ll show an example of Specialization use in a current state or change model, but largely I would advise you to be careful when using this relation type in models that are meant to describe a concrete (project (end) state or current state) reality. In other words, you can forget most of what has been written in this section and still properly model good ArchiMate models.

## 2.4 Products, Contracts and Value

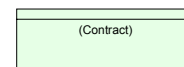
If you want to model the offerings of your organization to the outside world (anyone who uses what you produce, be it clients or regulators), a handy object is Product. A Product is a simple enough concept. It is an Aggregation of one or more services and (optionally) a Contract. It is Associated with a Value. It looks like this:



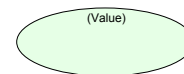
**View 26.** Product, Contract, Value and a Product’s Constituents

As you can see, the Product object type is a bit weird: it is part of ArchiMate’s Passive Structure (objects acted upon), but it Aggregates also Behavioral objects (the actions themselves). And it is a business-layer object, but it aggregates also an application-layer object. If I was a

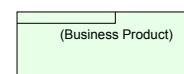
purist, I would say the object type is a ‘kludge’. Still, it can be useful. The new object types are:



This is the **Contract**. It represents the formal or informal agreement that covers the delivery of the service provided. This might be a specific Service Level Agreement or General Terms & Conditions.



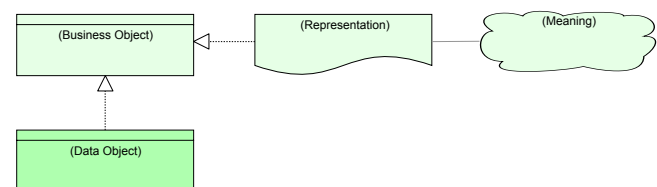
This is the **Value** of the Product. A Value is a pretty abstract kind of object in ArchiMate. It has wide-ranging use. It can be the value to the consumer, but also to the producer. It could be described in monetary terms, but it may just as well be described in emotional terms (e.g. an insurance service may be associated with the value of ‘economic security’ or ‘be able to sleep easy at night’. ‘Be able to’ since ‘sleep at night’ is the actual Business Process). In the standard view of the ArchiMate model, it is generally Associated with the Product (and this is also the case in the underlying meta-model as shown in the specification), but the actual ArchiMate definition of Value Associates it with the service and mentions the Association with Product only as ‘indirect’.



This is the **Product**. It is what you offer to the outside world. For a department of your organization, the Product may be something offered ‘internally’. It can be handy to make the link between your Enterprise Architecture and how management looks at the organization.

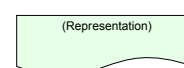
## 2.5 Representations and Meanings

Earlier we encountered the Business Object, the work horse of passive structure in your business layer architecture. And we saw how this business-level object could be Realized by an application layer Data Object. Take for instance the ‘Answer’ Business Object that was Realized by the ‘Word Document’ Data Object in View 6 on page 19. ArchiMate also has another way to Realize a Business Object, it can be realized by a Representation. A good example would be a print of the answer letter you are sending to your customer. The relation between a Business Object and its direct surroundings looks like this:



**View 27.** Representation and Data Object Realizing a Business Object

The new object types are:



This is the **Representation**. While the Data Object is the object in our application layer that represents the Business Object, the Representation is another way of representing the Business Object and it stands for a representation that can be shared with others, e.g. a print, a PDF file you send as attachment in a mail message, or a web-page.

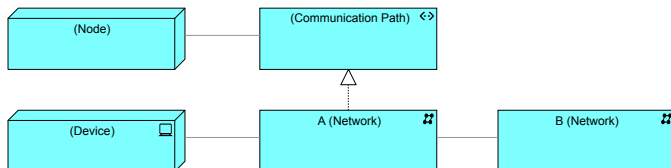
(Meaning)

This is the **Meaning** of the Representation. ArchiMate says it is the “information-related counterpart of a Value” and it “represents the ‘intention’ of a Business Object or a Representation”.

For Corporate Strategy Level Enterprise Architects, they can be useful, e.g. in the future state architecture, but in most practical circumstances (current state architecture, project architecture) they do not add a lot of value. Especially Meaning runs against Uncle Ludwig’s idea about what a meaning is, but that is a philosophical issue, not an architectural one, so I am not discussing it here. Representation is sometimes a useful link to the physical world, but the description ArchiMate gives with links to ‘RTF’ or ‘HTML’ overlaps with the IT-levels of your architecture. We’ll return to this issue later.

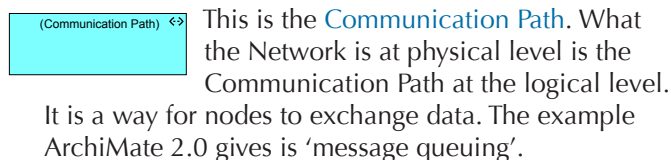
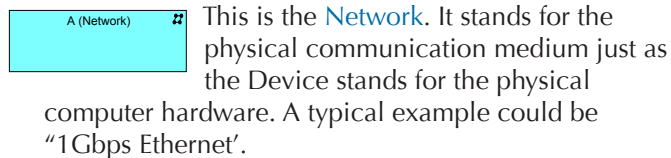
## 2.6 Network and Communication Path

ArchiMate offers two objects to model communication at the infrastructure level: Network and Communication Path. In a simple overview of use:



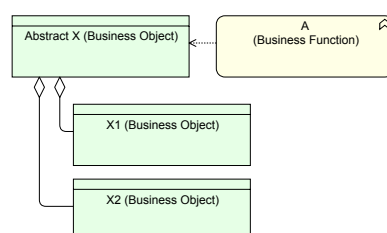
View 28. Network and Communication Path

The new object types are:



A Network Realizes a Communication Path. The other relations are all Associations. To be honest, I have never seen these used in practice. Maybe I haven’t seen enough practice or maybe they were thought of in a time that such aspects were still important in Enterprise Architecture work while they are now more or less a given like the air we breathe.

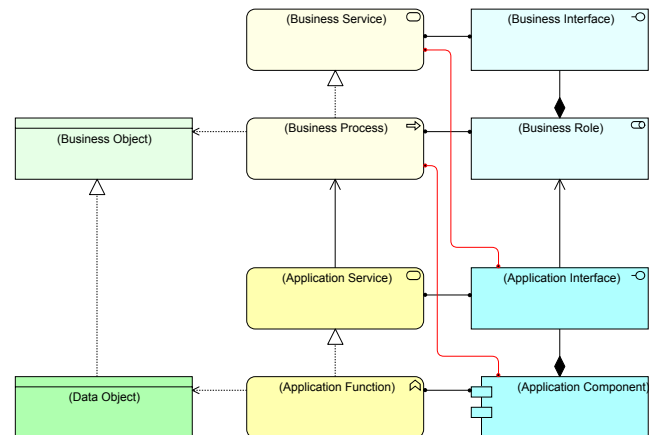
Network and Communication Path can be drawn both as ‘boxes’ with the association relation and also as connections, even though they technically are objects. Basically that would mean that you have to draw a Realization relation between two connections and my tool of choice cannot do that, so you’re not seeing it here.



View 44. Aggregation can be used as grouping

## 2.7 Automated Processes

So far, our landscape was based on a business process performed by people (actors fulfilling a role). But what if a process runs automated? ArchiMate has the following solution: If a process is run by people, a Business Role is Assigned-To the Business Process and a Business Interface is Assigned-To a Business Service. The Business Role *performs* the Business Process. If a Business Process is *performed* by an application, we draw an Assigned-To between the Application Component and the Business Process. And consequently, we also draw an Assigned-To between the Application Interface and the Business Service. Here they are (in red):



View 29. Automated Process

This way, we can model a business process that by itself performs a service. In Section 17.3 “Automated Processes” on page 104, we discuss this approach in more detail as part of a Chapter on proposed improvements for ArchiMate, for instance why we would or wouldn’t also model the application service as in the example above (we do).

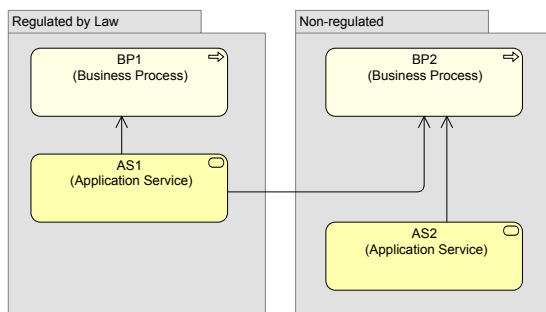
## 2.8 The Grouping Relation

In ArchiMate you can group objects visually in two ways: Nesting (see Section 1.8 “Nesting” on page 20) and Grouping. Nesting is in fact a way to draw *three* types of relations: Composition, Aggregation and Assignment. According to the standard you may *only* use Nesting for one of these relations (tools often allow more nestings than ArchiMate allows). But what if you want to visually group other objects? Suppose you want to group the different Business Objects accessed by a Business Function? You can for instance use Aggregation and create an abstract Business Object in your landscape as you can see in View 44. On the left of that view you see the actual relations, on the right the Aggregations are drawn using Nesting. It works if you can Aggregate and if you do not mind adding an abstract (non-existing) object in your model.

But what if you cannot aggregate? ArchiMate 2.0 has cleaned up the allowed relation table and for instance, you cannot aggregate a set of Business Roles under a Business Actor, as it used to be possible in ArchiMate 1.0. For this, ArchiMate has the **Group-**



ing relation. Which is a relation, but looks like an object with other objects nested like the gray boxes with labels in View 30.

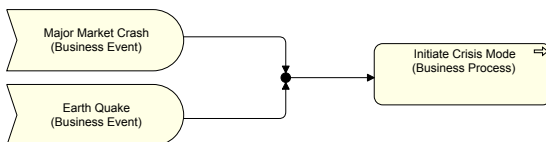


View 30. Grouping Relation

The Grouping relation is a catch-all kind of relation. It says more or less the same as if all objects in the box have something in common, in the example: legal status. So, in the left box, we find a Business Process 'BP1' and an Application Service 'AS1' that are regulated by law, and in the right box we find a Business Process 'BP2' and an Application Service 'AS2' that are non-regulated. Given the catch-all nature, you can think of grouping as a visual way to show Associations, but ArchiMate does not define it that way. Though Grouping is formally a relation in ArchiMate, it tends to be used in a pure graphical way to make certain views of your model clearer for the reader.

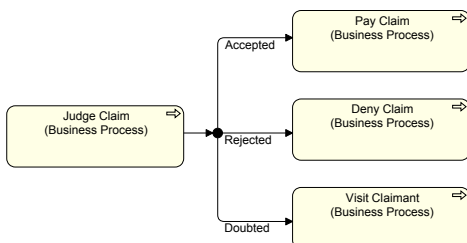
## 2.9 The Junction Relation

Finally there is the Junction relation. This is very special because it is a *relation* that can relate other (dynamic: Flow and Trigger) relations to each other. Here is an example:



View 31. Junction relation (join)

The big dot in the middle is the Junction. You can label the incoming or outgoing dynamic relations, e.g. in a case of a choice:

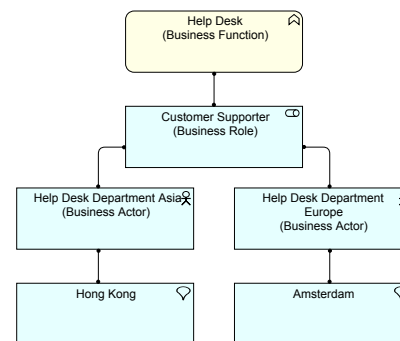


View 32. Junction relation (split)

The Junction is very generic. The ArchiMate 2.0 Specification suggests that it might be handy to extend it (ArchiMate is extensible) to And-Split, Or-Split, And-Join and Or-Join. But that is not part of ArchiMate itself. The tool I use comes by default with an And-Junction and an Or-Junction (not separated in splits and joins).

## 2.10 Location

The final ArchiMate object to explain is Location. This is new in ArchiMate 2.0 and it can be seen in an example in View 33.



View 33. Example use of Location object

This is the Location object. This object stands for a geographical location where something resides. You can use the Assignment relation to link it to several other object types to model where they are located.

## 2.11 The Complete Picture

ArchiMate is reasonably rich. There are 31 object types and 10 relation types in version 2.0, excluding the Motivation and Implementation extensions.

In View 45 on page 28 you'll find the overall picture of ArchiMate's object types and their relations. This is often referred to as the ArchiMate Meta-Model (note: Business Process, Business Function and Business Interaction take the same place and are represented by the Business Process icon in this view of the meta-model)

Page 28 is repeated on page 131. During the rest of the book, you might want to look back at the meta-model to check for things. If you keep your finger between that page and the one before it, it is always readily available.

## 2.12 Closing Remark

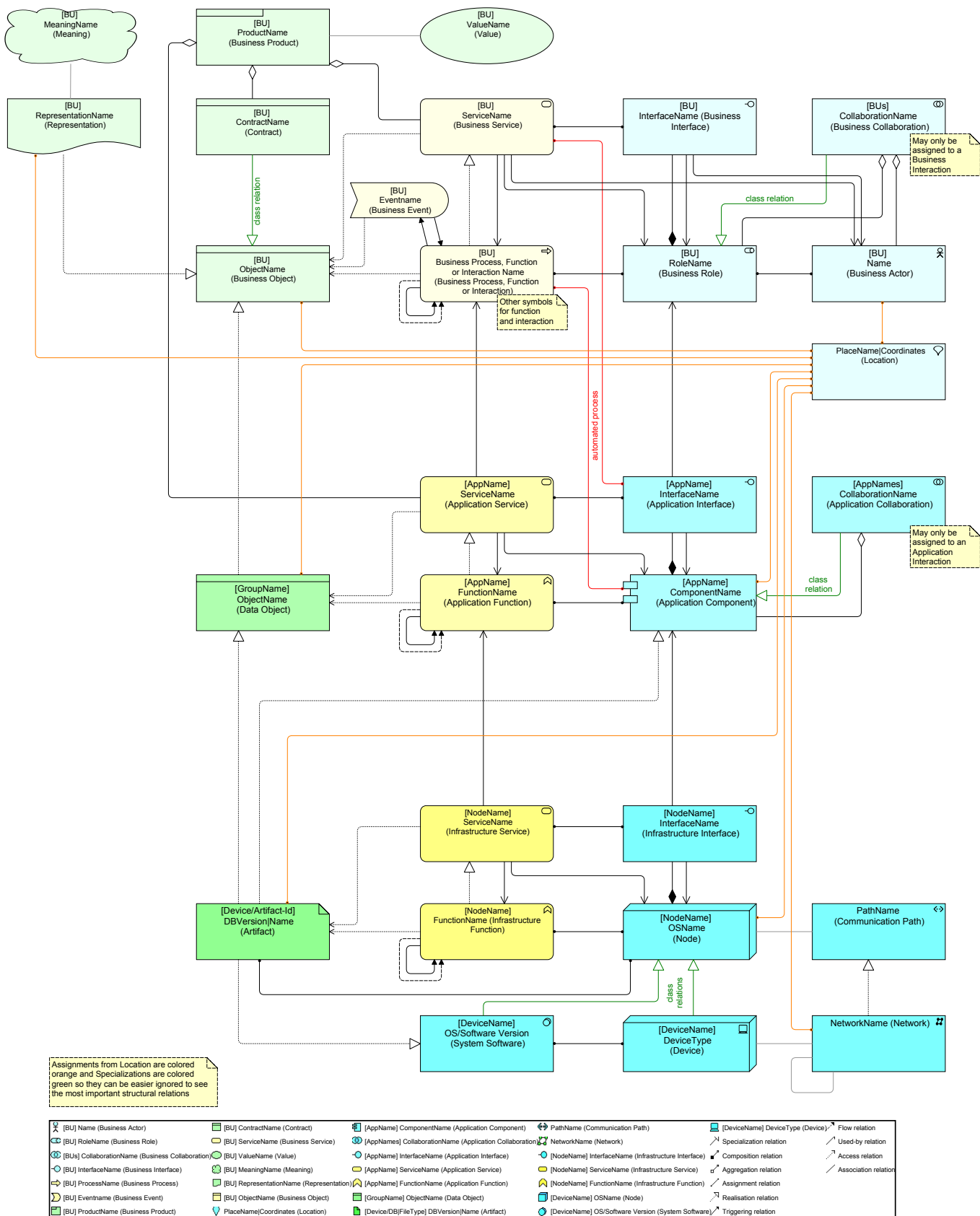
We have seen relations that look like objects (Grouping) and objects that may look like relations (Communication Path). If you get the feeling that ArchiMate is not 100% clean, I cannot fault you. But even with these issues (which we will visit later) I must stress that in my experience, ArchiMate is *extremely* usable. Creating Project Start Architectures based on detailed ArchiMate models has in my experience substantially reduced delays and unforeseens, having a detailed 'Current-State' model has seriously improved our control over and our reporting on our landscape to regulators.

When we started to use ArchiMate, we initially had many doubts about how the ArchiMate designers had set up the language. But, we did follow the rule that we would strictly stick to the meta-model and not change or adapt it in any way (even though our tool supported that). We did this for two reasons:

- Any future tool update might break our models or require a lot of work;

- Understanding something comes only after a lot of experience. Though we did sometimes dislike what we saw, we decided to distrust our (beginner) skills more than the skills of the language designers. This was wise: we learned to appreciate most of the choices the designers made.

If you focus on finding faults (something that comes natural for architects as they are always for the lookout of something that may break) you will. But in my experience, that does not affect the usability much. ArchiMate is very, very usable.



# 3. Derived Relations

## 3.1 Derived Structural Relations

So far we have included every concept, every object type and all connecting relations. ArchiMate, however, offers a mechanism to create ‘shortcuts’. The researchers, business users and students that created ArchiMate even offered a mathematical proof that their definition of the shortcuts are ‘correct’. Adding these shortcuts to the model was originally seen as extending ‘ArchiMate proper’ to a sort of ‘ArchiMate+’, but these shortcuts are so handy and useful that most practitioners (or teachers) do not really distinguish between the fundamental and shortcut relations anymore (which is a bad thing, as we will see). Add to that, that tools generally do not support the actual difference and it is easy to understand that the shortcuts are probably one of the most (implicitly) used but least (explicitly) understood aspects of ArchiMate.

Officially, ArchiMate calls these shortcuts ‘derived structural relations’ and I generally explain them by telling that the derived relations are a summary of a route that lies between two objects in a model. All tools I know allow you to model a summary without modeling the underlying ‘true’ route of objects and relations, but in the end each summary relation more or less assumes such a route with real objects and relations is there, even if you have not (or need not have) modeled them in your model.

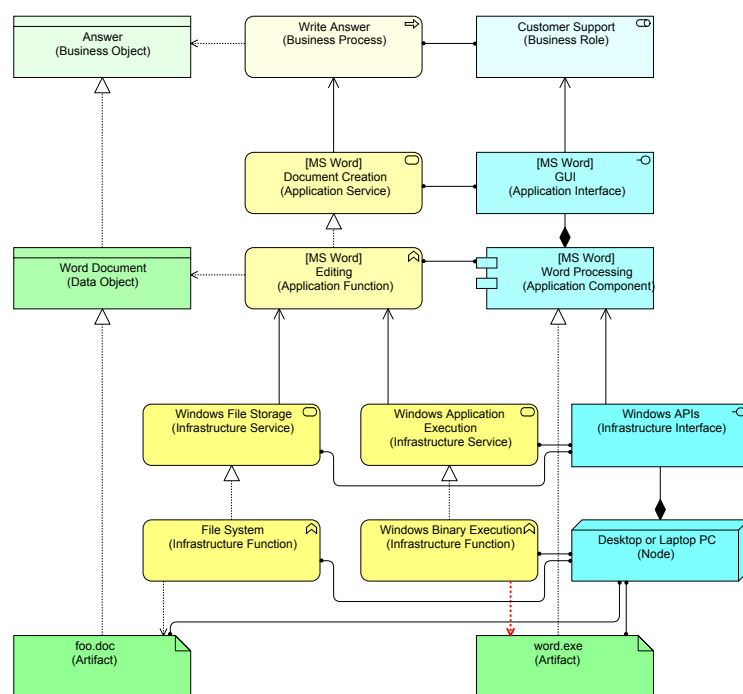
The best way to illustrate them is by using an already familiar example, the earlier used ‘write answer’ example. It is shown again in View 46 on page 29 for easy reference (ignore the redness of one relation this time).

Given this example, the questions we want to answer are like:

- What is the relation between the ‘Desktop or Laptop PC’ Node and the ‘Write Answer’ Business Process?
- What is the relation between the ‘word.exe’ Artifact and the ‘Document Creation’ Application Service?
- What is the relation between the ‘Desktop or Laptop PC’ Node and the ‘Answer’ Business Object?

For this, ArchiMate comes with the following procedure:

- Find a route from one object in the model to another following the *structural* relations between them. There is a additional requirement: a route is only valid if all relations followed are followed in the *same direction*. All relations have a direction, except Association which is bidirectional (Assignment was also bidirectional in ArchiMate 1.0).
- Every structural relation has a *strength*. If you have a valid route, the derived relation between both ends of the route is the *weakest* relation found on the route like the weakest link in a chain. The strengths are (from weak to strong):
  - Association (bidirectional))
  - Access (direction: always from behavioral object to passive structural object, independent from



View 46. Write Answer Process, supported by MS Word and a Standalone PC

arrow which depicts read or write. That the direction may be opposite to the arrow is a pitfall when starting with ArchiMate.)

- Used-By
- Realization
- Assignment (in each layer from actor to behavior and in the infrastructure layer from Node to Artifact, bidirectional in ArchiMate 1.0)
- Aggregation (direction: from parent to child)
- Composition (direction: from parent to child)

Using this procedure, we can answer the questions above:

- From the ‘Desktop or Laptop PC’ Node, via Assignment (strength: 5) to ‘Windows Binary Execution’ Infrastructure Function, via Realization (strength: 4) to ‘Windows Application Execution’ Infrastructure Service then via Used-By (strength: 3) to the ‘Editing’ Application Function then via Realization (strength: 4) to the ‘Document Creation’ Application Service then via Used-By (strength: 3) to the ‘Write Answer’ Business Process. The weakest relation encountered is Used-By, so that is the relation between the PC and the Business Process: *the PC is Used-By the Business Process*, which makes good sense.
- The ‘word.exe’ Artifact Realizes (4) the ‘Word Processing’ Application Component, which is Assigned-To (5) the ‘Editing’ Application Function which Realizes (4) the ‘Document Creation’ Application Service. The weakest is Realization, so *the ‘word.exe’ Artifact Realizes the ‘Document Creation’ Application Service*. Which also makes sense.
- If we take the route from (a) we need just one extra step: from the ‘Write Answer’ via Access (strength: 2)



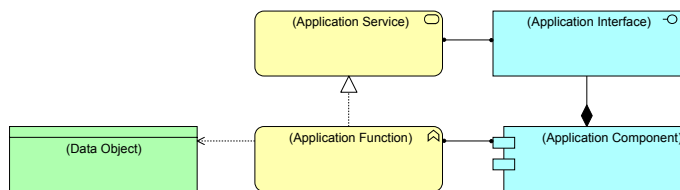
to the 'Answer' Business Object. The weakest of that route is now Access, so *the PC Accesses the 'Answer' Business Object*.

The Open Group's ArchiMate Forum's decision to make Assignment unidirectional in ArchiMate 2.0 solved the problem that quite a few nonsense derivations were possible. In ArchiMate 1.0 it would for instance enable the route from the 'foo.doc' Artifact via the PC to the 'Windows File Storage' Infrastructure Service with Realization as the resulting relation, or in other words: 'foo.doc' Realizes Windows File Storage. Hmm, I think not.

But still, sometimes, multiple shortcuts do exist. For instance, there is an alternative answer to question (c): From the PC Node via Assignment (strength: 5) to the 'foo.doc' Artifact, then via Realization (strength: 4) to the 'Word Document' Data Object and via another Realization to the 'Answer' Business Object. The result is: the PC Realizes the 'Answer' Business Object.

In other words: the PC Accesses the 'Answer' Business Object and it also *Realizes* that same 'Answer' Business Object. Here it is clear that we are looking at two aspects of this PC, it both executes the application and it stores the data. If the data were to reside on another server, we would not have both routes. So, the two routes are in fact a true aspect of the situation: the PC does both and so both relations dutifully appear. No problem.

So, all is well? It gets a little more complicated. Have a look at the Basic Application pattern again in View 34.



View 34. The Basic Application Pattern

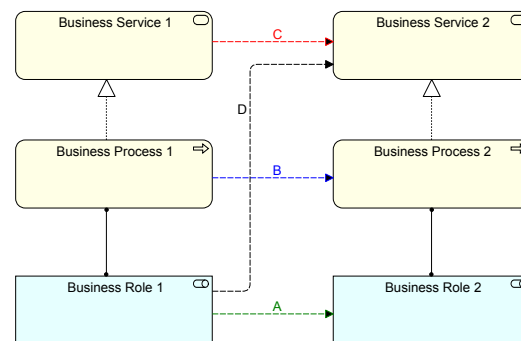
If you go from Application Component to Application Service, you can follow two routes. The first route (via the Application Interface) leads to Assigned-To as the (weakest) derived 'summary' relation. But the one via the Application Function leads to Realization as the (weakest) derived 'summary' relation. ArchiMate does not tell you which one to take if two possible routes exist. That is OK, as we saw in the previous example, when two routes stand for two different aspects. But in this case, either derived relation says something about how you view the concept of Application Component. If you choose Assigned-To, you keep to the strict separation of active component and its

behavior. If you choose Realization, you look at the Application Component from a more behavioral point of view. We get back to this confusing issue when we discuss the language and look at possible improvements in Section 17 "Proposed Improvements" on page 103.

What all of this so far implies is that derived relations are useful to make summary views of a complex landscape, but there are risks involved in using them without (or mixed with) the underlying reality. The risks we seen so far are still benign. But there are some pitfalls we will see below.

## 3.2 Derived Dynamic Relations

Quite a bit simpler, but since ArchiMate 2.0 there is also the possibility to create derived relations from *dynamic* relations. Take a look at the following example:



View 35. Derived Dynamic Relations

Suppose we start with the Flow relation B, between 'Business Process 1' and 'Business Process 2'. ArchiMate says:

- You may move a begin or end point of a dynamic relation to an object that is Assigned-To the current object.
- You may move a begin or end point of a dynamic relation to an object that is Realized by the current object.

Moving both begin and end points simultaneously, we can turn Flow B into Flow and turn Flow B into Flow C. Note that we cannot turn Flow C into Flow B or A or turn Flow A into Flow C

You can of course move only one of the begin or end points. E.g. from Flow relation B, you can derive a Flow relation D between 'Business Role 1' and 'Business Service 2'.

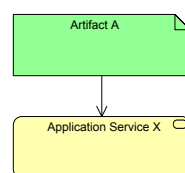
The same is true for the Trigger relation and the same derivations can be made on the application layer and — since ArchiMate 2.0 — on the infrastructure layer.

## 4. Beginner's Pitfalls

### 4.1 Pitfalls of Derived Relations

I once encountered the snippet in View 36.

The modeler intended to model a file that was accessed by an application. He used an Artifact for the file (which makes sense), but his tool did not allow him to draw an Access relation from Application Service X to the Artifact. His tool suggested, though, that the Used-By was possible.

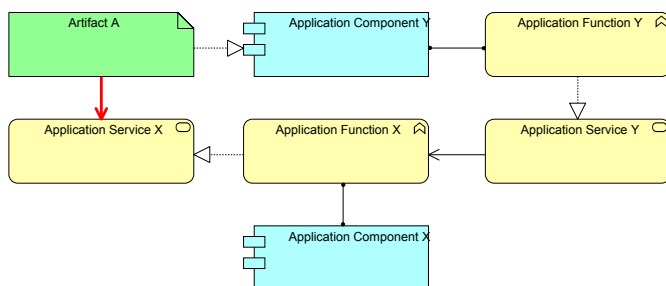


View 36. Artifact Used-By Application Service?

Having an Access relation from Application Service to Artifact is not in the core metamodel. Having an Access relation from Application Service to Artifact as a derived relation is also not possible in ArchiMate as it requires traveling a route with relations in opposite directions (Realization from Artifact to Data Object and Access from Application Service to Data Object).

But his tool did allow him to model a Used-By from Artifact to Application Service. The Architect just used what was allowed, and from a human language point of view, it made sense: “the file is ‘used by’ the application service”. If you say this to a fellow human being, he or she will know what you mean. So, he assumed that all was well and driven by an understandable desire to be productive carried on.

But in ArchiMate, this is not what it means because ArchiMate’s Used-By is not equivalent to natural language’s ‘used by’. Since this Used-By is not a core relation, it is some sort of a derived relation. So, the question is, what route lies behind this derivate? What hidden assumptions are there that make this relation possible? Here is possible expansion of what his derived Used-By relation meant (his original relation in red):



**View 37.** Artifact Used-By a Business Service: the hidden objects

As you can see, the Used-By from Artifact A to Application Service X was possible, because an application can use *another* application (and not so much data), and that *other* application is then Realized by our Artifact A. He thought he modeled the use of data but he did model the use of another (hidden) application.

If the language allows the relation, it is not certain that the relation actually means what you think. That shortcut leaves intermediary structure out, and sometimes (as shown here) that structure in ArchiMate must be something that is not what you intended to show. This is one reason why I generally say that it is best to keep as close as possible to the core relations in the meta-model and not use derived relations too much when modeling. You can of course use them for reporting, but using them in modeling is risky.

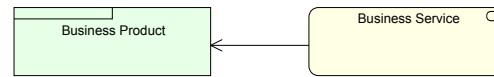
**Pitfall 1.** Derived relations may not mean what you think.

## 4.2 Don’t trust the language or the tool blindly: know what you are doing

A tool sometimes offers you to draw a ‘default relation’ between two objects. If you draw a relation between an Application Component and a Business Process, it may prefer the core relation Assigned-To above the derived relation ‘Used-By’. I have seen quite a few beginner models that had Assigned-To relations between Application Components and Business Processes while it was *not* an automated business process. They meant to write that an Application Component was Used-by a Business Process (a derived relation and also what they

meant to model) but ended up drawing an Assignment and thus effectively writing that the application automatically performed the Business Process.

Another beginner modeled View 38 (in the ArchiMate 1.0 days):



**View 38.** This was possible in ArchiMate 1.0

For this one I could not find a derived route at all. Here, it turned out, the ArchiMate 1.0 specification did allow it (it was in the table that has all the possible relations and derived relations). This was a leftover of earlier versions before ArchiMate was officially standardized. It was never removed from the table and dutifully implemented by the tool builder. Sadly, though, it was an error in ArchiMate 1.0. The error, by the way, has been fixed in ArchiMate 2.0, so decent tools in recent versions will not allow you to draw this relation anymore.

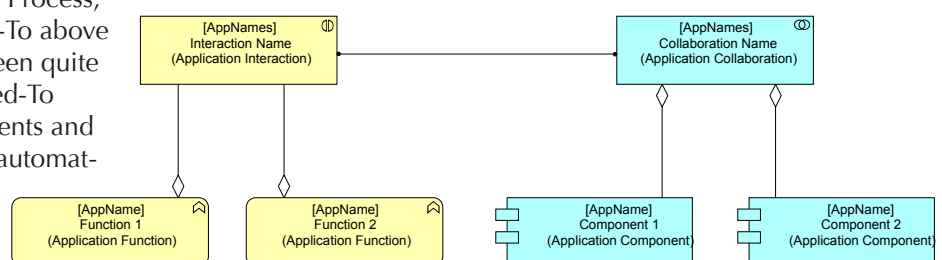
**Pitfall 2.** Not checking if your relation makes sense in the meta-model

And finally, I once modeled an Interaction and a Collaboration. To my surprise, the tool allowed an Aggregation from Interaction to Function, but not the other way around as I wanted. It can be seen in View 47. Now, this is not exactly how ArchiMate defines the Interaction objects (Interaction objects are not defined as an Aggregate of functions/processes, they are just the behavior of a Collaboration, which is an Aggregation (of roles or application components, recall Section 2.1 “Collaborations and Interactions” on page 23).

Whatever I did, I could not get the Aggregation relations on the left the way they are on the right, the way they make sense when you want to model this detail. It turned out, the tool makers had forgotten to implement the meta-model relation between Interaction and its constituent Functions and when you tried to create it, it found an other relation that was possible because of Specialization: because Interaction and Function are both Specializations of an underlying behavioral object in the meta-model, and because any type of object can Aggregate its own kind, it was possible to Aggregate an Interaction under a Function.

**Pitfall 3.** Trusting your tool blindly.

Lastly, ArchiMate (and several tools) support ‘viewpoints’. Viewpoints are in fact subsets of the ArchiMate meta-model (the subset may include derived relations). A viewpoint may restrict which types of objects you may model and what relations are allowed between them. They are a sort of patterns, but not quite, as they do not so much pre-



**View 47.** Not quite what ArchiMate intended,

scribe *how* to model but only offer constraints in *what* you can do.

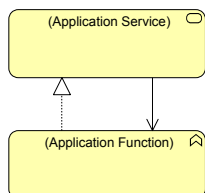
I use 'view templates' for modeling (see Section 15 "Construction & Use Views" on page 97). A tool will constrain your use of objects and relations in an ArchiMate Viewpoint and not being able to create a relation may not be because it is not allowed, but because it is not allowed in the active Viewpoint.

**Pitfall 4.** Using a viewpoint without knowing that you are.

### 4.3 Misunderstanding the standard meta-model diagram

I have seen beginners often just copy the relations they see between object types in the basic meta-model (View 45 on page 28). This is generally fine, but there are some snags.

A common snag is the way the standard meta-model shows the relation(s) between a function/process and a service. This looks like View 39.

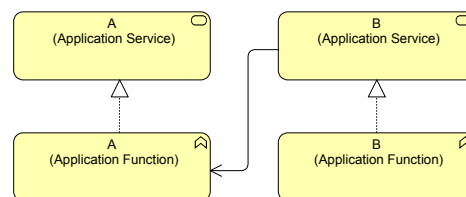


**View 39.** How Function and Service are displayed in the Meta-model

What the view illustrates is that 'a' function can realize 'a' service and that 'a' service can be used by 'a' function. But in real modeling, we model generally not 'a' service or function. We model specific services and functions.

So, when a beginner models something like View 39 as the relations between a specific function and the service that it provides, he actually says the function also uses the service it provides: it uses itself. Now, as a way of modeling recursion, this could be OK, but that is generally not what is meant (we generally do not model this kind of technical details in Enterprise Architecture). In short: the pattern of View 39 should generally *not* appear in our models.

What the above snippet illustrates is that *an* object of the type Application Function can Realize *an* object of the type Application Service, and that *an (other)* object of the type Application Function can use *an object* of the type Application Service as illustrated in View 40.



**View 40.** How the function/service relations in the meta-model are intended.

The Realize and Used-By relations are between the same types of object, but not between the same *objects*. And of course, the same is true at the business and infrastructure levels.

**Pitfall 5.** Copying relations in the meta-model to your model blindly.

## 5. Some Noteworthy Aspects of ArchiMate

### 5.1 Introduction

ArchiMate started out as a University Institute project, supported by a couple of large organizations. Its initial version was finalized in 2004. In 2009 it was adopted as an open standard by The Open Group. ArchiMate is — as far as I know — the first *Open Standard* for Enterprise Architecture modeling (if you do not count UML as an EA modeling language). Enterprise Architecture is not new and quite a bit of modeling has been done, most of it not in ArchiMate.

People create models for projects, often just as a set of images in an 'architecture' document. Sometimes, some modeling for projects is done in UML, but most of the time, you will look at some non- or semi-standardized use of boxes, arrows, dotted lines, nesting, etc., generally some sort of free-format graphical tooling will be used like Microsoft Visio for Windows or OmniGraffle for Mac or worse: Microsoft Powerpoint. One of the 'nice' aspects of such modeling is that it is often ambiguous enough for all stakeholders to see their own preferred reality in it. A more vague and ambiguous approach enables this often 'politically' expedient modeling. Everybody is happy, that is, until it turns out there is a problem somewhere during a

later phase of a project. In my experience, people do not understand detailed Visio images either, but they seldom have to understand them anyway, so everybody is happy. When ArchiMate arrives, suddenly people are confronted with views that they have to agree on as a definition of sorts, and suddenly they have to understand what the image says. And the result is that some of them will fight adoption of the language. They want to stick to the old simpler way.

It gets even worse with the modeling of your current state. Once every few years, the lack of insight in the existing landscape will cause an effort to create an overview. People work for months at unearthing the situation, and generally in the end you will have a few large posters with boxes and lines (and a legend to explain what every box type, arrow type and color signifies). The model is never maintained and after a year, it becomes so outdated that it becomes pretty useless for anything but a general introduction. Now, also in ArchiMate, this can happen, if you just draw your visuals in ArchiMate without the support of tooling that can keep thousands of objects and relation in a single coherent model.



The fact that a single coherent model is useful is of course clear to many. So, sometimes modeling is done in Enterprise Architecture tooling. These tools often have their own internal proprietary model, a model that also comes with a certain philosophy on what Enterprise Architecture is. These days, some of these tools will have ArchiMate as an option, often (more or less successfully) implemented on top of their own proprietary language. I'll say a bit more about tooling in Chapter "Tooling" on page 115.

Modeling in ArchiMate brings the advantage of a fixed syntax of your models with reasonably clearly defined object and relation types. Modeling with a tool brings you the possibility to have those different views on what you are modeling to be actually related 'under water' and it brings more possibilities to actually maintain your models. Modeling in ArchiMate also brings modeling that is independent of your modeling tool: if you ever decide to move it is possible to another tool that supports ArchiMate. In the worst case the migration cannot be automated and migration is a lot of hand work, but it is possible.

In Chapter "Discussing ArchiMate" on page 103 I will discuss some of the areas where I think ArchiMate could be improved. In this chapter I'll say a few things about its strengths and some weaknesses I have no suggestions about. But, though I have my criticisms, *my overall conclusion is that the language is very practical in actual use*. I would not have written a book about it if I were not convinced that ArchiMate can be extremely useful for Enterprise Architecture work.

In the final document of the original university project, the authors write:

*During the initial design of the ArchiMate language no explicit attention was paid to any desired formal properties of the metamodel itself. Emphasis was put on the applicability of the language.*

And this is also what has resulted: a very *usable* language, which is not perfectly logical. This sometimes offends architects who desire a formal, certain language which can be used without interpretation (or so some architects think). ArchiMate offers a 'rational' way of looking at Enterprise Architecture, but it does not offer a 'perfectly logical' way. Later versions of ArchiMate have improved the language and some original choices leading to severe problems (like the original bi-directionality of the Assignment operator) have been fixed. But it still remains noticeable here and there that the language never was based on a formal model.

## 5.2 Separation of Actor and Behavior

At the business level, separating process (behavior) from role (actor) is common practice. There is some confusion with respect to a business function (as will be discussed in Section 10.2 "Business Function or Business Process?" on page 72).

Some approaches see a function as behavior embedded in some sort of actor-like object, you can recognize it in the language used: a function *is* behavior (as in ArchiMate) versus a function *does* something (function *performs* behavior, it seems to be an actor of sorts). ArchiMate is a language where the separation is fundamental and the

separation has been replicated in all layers. This clarity in the meta-model really helps to make good models, but it is important to stress that at every layer, the actor and its behavior are fundamentally inseparable. You can choose not to *model* one and use derived relations to pass them by, but they are assumed to be there.

The split often is confusing for those coming from ways of thinking that do not have that split, e.g. Process Modelers who do often not use the function concept at all (for more depth, see: 10.4 "The 'End-to-end' Business Process" on page 75) or Software Architects where (like in mathematics) a function is both structure and behavior in one.

There is another misinterpretation I've come across. If you say 'passive object' to people, they sometimes interpret that as 'not taking initiative', as 'reacting only'. This I encountered for instance when a colleague first was told about the interface concepts in ArchiMate. He found it illogical to say that an interface is an active object. Because an interface does nothing by itself, it needs to be used before it does something. In his use of the words active and passive, that meant an interface is passive. But ArchiMate does not mean 'reactive' by the word 'passive' and it does not mean 'taking initiative' by the word 'active'. In ArchiMate terms:

- **Active** means *capable* of performing behavior (either acting or reacting);
- **Passive** means *incapable* of performing behavior.

Behavior is what links active to passive objects: Active objects are Assigned-To behavior and behavioral objects Access passive objects. The passive ArchiMate objects are philosophically speaking 'objects' and the active ArchiMate objects are philosophically speaking 'subjects'. It is not wrong what the colleague said, it is just not how ArchiMate intends the terms 'active' and 'passive'. For completeness, here are ArchiMate's official underlying core definitions of active, behavior and passive:

*An active structure element is defined as an entity that is capable of performing behavior.*

*A behavior element is defined as a unit of activity performed by one or more active structure elements.*

*A passive structure element is defined as an object on which behavior is performed.*

And service and interface are defined as:

*A service is defined as a unit of functionality that a system exposes to its environment, while hiding internal operations, which provides a certain value (monetary or otherwise).*

*An interface is defined as a point of access where one or more services are made available to the environment.*

## 5.3 One View Type

One of the beautiful aspects of ArchiMate is that it is a true Enterprise Architecture language. ArchiMate has been designed to model the interrelations across *all* layers in your Enterprise Architecture: from Business via Applications down to Infrastructure and vice versa. Though tooling of-

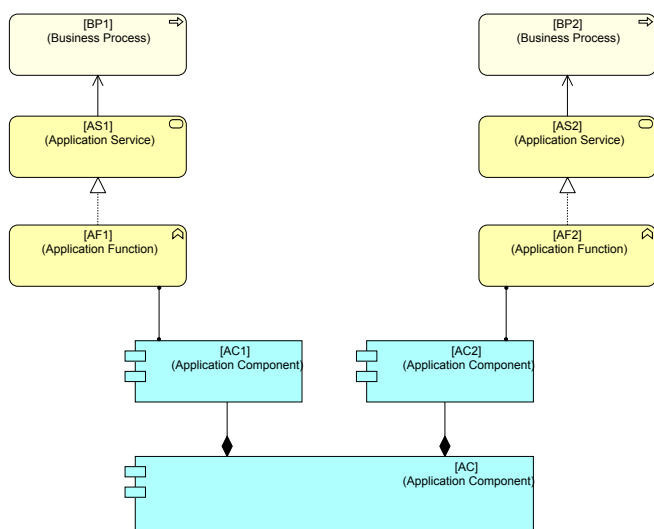
ten offers you all kinds of different view types, ArchiMate itself only has one single view type: the entire Enterprise Architecture. ArchiMate *Viewpoints* (which I do not strictly use) restrict what is in an ArchiMate view, but it is not a fundamentally different view.

At the software level, UML with all its different kind of views on the same objects and classes might be more precise, to model the whole shebang, ArchiMate does a very good job.

## 5.4 On the direction of structural relations under the assumption of ‘worst case’

In ArchiMate 2.0 the structural relations all have a single direction except Association, which is bidirectional. This direction plays an important role in calculating the derived relations between objects that are indirectly related in a model.

The idea behind it seems logical. Take the example in View 41.



View 41. Example used for discussing ‘worst case’ versus ‘happy flow’ assumptions about dependencies

If you look at it, it is logical to have a derivation mechanism that makes Business Process BP1 independent from Application (sub)Component AC2. After all, Business Process BP1 uses only Application Function AF1 and not Application Function AF2. But the question is, what do we mean exactly by dependence? Business Process BP1 is clearly not *meant* to depend on Application Function AF2, but does that mean it is independent? Suppose Application Function AF2 has a fault and it breaks the Application Function of Application Component AC and as a *consequence* Application Function AF1? In that case, there is a relation between Application Service AS1 and Application Function AF2, even if the relation is not *meant* to play a role and even if the relation from Application Component AC2 to Application Component AC has the opposite direction as the one from Application Component AC1 to Application Function AF1.

Or that interesting example of an Infrastructure Service that does not depend on what Application Function or Infrastructure Function uses it, except of course when the Application Function or Infrastructure Function overwhelms the resources of the Infrastructure Function which

for instance happens when a Denial-of-Service attack hits your web servers, but it can also happen in normal cases. Suddenly, in that ‘worst case’ there is a dependency from ‘user’ to ‘used’.

We can see that how you interpret ‘dependence’ has to do with the scenarios you take into account. Look at ‘happy flow’ only, and it makes sense that for instance Composition and Aggregation have a direction. But look at ‘worst case’ (a natural way of looking from an architectural point of view) and the Composition and Aggregation relations will link objects the other way too.

So, if you look at ‘worst case’ instead of ‘happy flow’, all structural relations become bidirectional. Going back to the example above: there seems no way Business Process BP1, Application Service AS1, Application Function AF1 and Application Component AC1 can depend on Business Process BP2, because whatever breaks in Business Process BP2 cannot influence Application Service AS2. But if BP2 overwhelms AS2, AS1, AC1 and thus AC, BP1 will be affected. BP1 then depends on BP2.

Maybe in a well-designed model you could give properties to the relations about their influence strength in both directions and then calculate the strength of dependence across a model (but now I am dreaming).

## 5.5 On the limitations of derived relations

So, relations in ArchiMate have a direction and this direction plays a role when creating valid derived relations.

The directionality of relations was required during the development of ArchiMate because of the way derived relations were set up mathematically. In that set up, bidirectional relations (Assignment and Association) were for the sake of analysis replaced by two unidirectional relations.

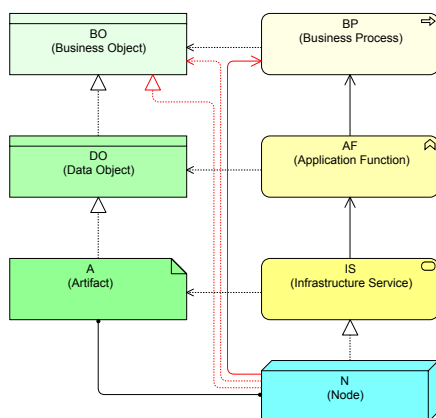
In ArchiMate 2.0, the bi-directionality of the Assignment relation was replaced by a unidirectionality. This is a good thing and it removed quite a few nonsense derived relations. The ones that made sense (like a Business Service being Used-By an Actor) were added explicitly to the meta-model.

If I look at the relations and their direction and I want to have some sort of criterium to be able to decide on making summaries and which combinations should and should not be allowed, I end up choosing the concept of ‘dependence’. If object A depends via relation R on object B, but B does not via the same relation depend on A, then the relation from B to A is unidirectional. The clearest example is the Used-By relation: if an Infrastructure Service X is Used-By an Application Function Y, Y depends on X, but not the other way around. For the Infrastructure Service X it is totally irrelevant if it is actually used or not, it does not depend on being used (but as we saw above: this is not quite true, as real-world Denial-of-Service attacks prove). But — in a simplified design-sense — for Application Function Y, life stops when Infrastructure Service X is not available and not the other way around.

Take also, for instance, the Access relation. It has a direction that runs from the behavioral element to the passive element. Now, I can understand that in part: a Data Object that is created or deleted by an Application Function depends for its existence on the work done by the Appli-

cation Function. It also fits with subject-verb-object like structure of ArchiMate. But a Data Object that is read by an Application Function does not depend on it at all, it is the other way around: if the Data Object is not there or broken, the Application Function may fail. The object may influence the subject, as in “The broken tile caused him to fall”. In natural language, objects without behavior may be used as causes (become active). In ArchiMate, passive objects are not like that.

Or think of a Business Process that requires access to a certain document. The document is a Data Object that is Realized by a file Artifact and that itself Realizes a Business Object (e.g. a letter). The file Artifact resides on a server (Node). When you look at that situation, there is nothing weird or ‘wrong’ with thinking that the Business Process Accesses the Artifact and your business people will in reality really think that way. They think in files, not in abstract Business Objects. But the derived relation is *not* allowed because of the direction of Realization. You can create derived relations from Infrastructure to Process (a few are illustrated in View 42), but *not* the other way around.



View 42. From Node to Business Layer

One of the things that initially attracted me to ArchiMate was not only the reasonable meta-model that is its foundation. It was the promise of derived relations. To have a model where you can have all the necessary details for a decent analysis as well as views where you have summarized the complex views in simpler ones. And both views are correct and come from the same underlying structure.

I must admit, the official ArchiMate derived relations have lost a bit of their attraction for me. There are too many problems where you as an architect know that the language does not allow you a (derived) relation, but where you know it makes sense to have one. In my practice, it turns out the official derived relations have a very limited use. They are too limiting in many situations. It turns out that depending on the patterns you model with, you develop the need for quite different ‘derived’ relations.

I also have not seen a tool yet with decent support for derived relations. The original ArchiMate project created them as an extended set of relations on top of what is in the (powerful) basic meta-model. And they provided a mathematical proof that if they were defined as they were defined, the new set of ‘basic + derived’ relations was again a closed set. The current standard actually says they are as proper relations as the core relations. It also is being

presented in courses and educational material. I haven’t seen much material that goes into the limitations of this approach.

The use of derived relations in a model creates a lot of new analysis routes in models. And if you use tooling, the number of possible routes quickly overwhelms your effort to keep your analysis viewpoints (often some sort of viewpoint logic) working. This is why I advise to stick as much as possible to the basic meta-model and *only* use derived relations in the creation of your *patterns* (which we will see in Section 7 “Patterns”). That way you create a stable subset of all the possibilities that makes analysis of your models achievable. I advise against using too many derived relations other than those of your patterns.

## 5.6 Why two types of software?

ArchiMate has two types of objects for software:

- At the Infrastructure Level, there is System Software, which is Assigned-To an Infrastructure Function which Realizes an Infrastructure Service;
- At the Application Level, there is the Application Component, which is Assigned-To an Application Function which Realizes an Application Service.

Now, if you know a bit about computers, you know that this distinction is technically nonsense. After all, — with the exception of the operating system — software is software, whatever its use. That Relational Database System that you model at the infrastructure layer (e.g. Oracle, SQL\*Server, DB2, Postgres) is technically just another application. So, an IT-engineer will probably initially balk at ArchiMate’s separation of software in two types, where it is also can be a matter of taste (or confusion) how you model something. Take for instance the spreadsheet :it will be shown in Section 7.3 “Modeling Spreadsheet Applications” on page 46 that it can be modeled in two ways: both at the application layer and at the infrastructure layer. That seems stupid, why should there be such confusion possible in the language?

But if you look at the reality of Enterprise Architecture, the discussion is *not technical*. It is about modeling for the enterprise how business, applications and infrastructure together (should) work. And when you take the *business perspective* and not the technical perspective, it is quite clear that such a mission-critical application which has all that specific business process oriented logic embedded in its code is quite something different from that generic database that is needed to support it. If you look at the separation as ‘business-specific’ (i.e. containing business logic) versus ‘generic’, then the separation becomes understandable. Hence, the database is System Software, but the business-specific logic that is written inside it should be modeled as an Application Component, which is realized by that same database. As will be shown in the Application Deployment Patterns in Section 7 “Patterns” on page 44, it can become quite complex to create patterns for modeling modern architectures like three- (or even four-) tier application architectures and modern inventions like SaaS. On the other hand, you have to invent the patterns only once, you can use them many times.



For sake of argument, we could try to imagine if we were to try to have only one software type in ArchiMate. The System Software at the Infrastructure would become (the equivalent of) an Application Component. Then, either the Application Function should be able to realize an Infrastructure Service (which kind of makes the Infrastructure Service a clone of Application Service) or the Infrastructure Service realized by Software should become an Application Service (which means you have in fact software service versus hardware service). This leads to all kinds of problems with the business perspective on Enterprise Architecture. We either have application-layer objects in the infrastructure layer, which is confusing, or we have to limit infrastructure to hardware only. But the latter is also problematic because you can have many infrastructural components that can be Realized by both hardware and software. A couple of examples:

- A virtual hard disk in memory (RAM Disk)
- Software RAID
- A network filter built from a general purpose computer with two network interfaces and dedicated software

I think especially this latter problem is what makes a single type of software type in ArchiMate maybe logically superior, but in terms of a business perspective (and a usable view on Enterprise Architecture) inferior. From a technical perspective the meaning comes from how it is *implemented* (is it a Device or an Application Component?) but from

a business perspective the meaning comes from its *use* (is it business-specific or generic functionality?). I think for the usability of ArchiMate for Enterprise Architecture modeling it is a good thing ArchiMate has the (technically flawed) concept of two types of software. Uncle Ludwig wins every time...

## 5.7 There is more than one-to-one

An Artifact can Realize an Application Component. It can also Realize a Data Object. One data object can even Realize both. As will be described in more detail in Section 12 “Secondary and Tertiary Architecture” on page 84, realizing both will lead us to more complete thinking about our Enterprise’s Architecture. Or the same Artifact can Realize both active and passive objects, as will be shown for instance in 7.3 “Modeling Spreadsheet Applications” on page 46. ArchiMate is very powerful, because of that, but it might take some getting used to.

Sometimes, concepts from the real world you are modeling will end up in quite different ways in your model. This has not specifically to do with ArchiMate, but ArchiMate does have support for it. A good example is a customer. That customer is both an Actor in your Enterprise Architecture, but it might *also* be a Business Object. ArchiMate can let you do both and so catch the two quite different aspects of your customer (what he does and how he is represented inside your business) very well.

This page intentionally not left blank

# Patterns & style





# Style & Patterns

## 6. Aesthetics (Style)

### 6.1 Why aesthetics matters

If your views are simple, with only a few objects and relations, it will not be difficult for people to read them. But soon you'll find the need to create larger, more encompassing, views. And when you start having several tens of objects and many relations in a view, you'll find that it is hard for people to read them. It will even become hard for yourself if you get back to that view a few months later. It gets worse, when the view is a mess, visually.

Modeling with ArchiMate in that sense is not so different from writing ordinary text. Though in the end it is the conceptual that matters, layout can have a devastating effect on the readability of text. The use of a lot of different fonts, sizes, styles, colors, etc., creates a chaos that overwhelms the senses and makes it difficult to get at the concepts. And that is exactly what style does: when style is good it becomes invisible, at its best it even helps making things clearer. The main task for visual style — when used to improve readability, that is — is to become as little a distraction as possible.

When the Apple Mac ushered in the area of graphical computers in the 80's and formatting of text became for the first time something everybody could attempt, the first consequence was that the world was confronted with a lot of very ugly documents with disastrous visual styles and layouts. People tended to use every effect that was available in a single document. You see the same often when children start to create their first documents on computers. The same thing happened when the web went graphical in the 90's, you might remember the web sites with a lot of blinking, distracting, banners, for one.

In terms of Enterprise Architecture modeling we're — as I'm writing this — at the same 'childish' stage: most views look very ugly, are chaotic in their layout and style and in the end, are much less usable than they could be.

So, in this section, I'm going to give some simple suggestions to

improve the readability of your views. Note: architects, especially the ones that tend to create the more complex models, belong to the somewhat smarter demographics and as such also on average somewhat to the more 'lazy' demographics. Laying out your views cleanly takes some energy. Keeping the views well laid out when you change them even more: add one object and you might well up having to rearrange tens more. So be prepared to put some time in this. In my experience: it repays itself many, many times over.

The basic idea behind a good visual style is that the view should be 'easy on the eyes'. That means it should be 'quiet'. And next to that, you need some ways to fight ambiguity and you need some redundancy. To give an example, have a look at View 77 on page 40, it contains the ArchiMate meta-model in an unreadable layout. The readable layout was View 45 on page 28.

### 6.2 Arranging relations

The first thing to do is:

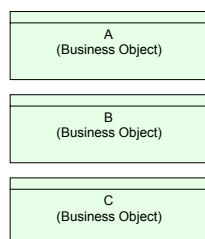
**Style Guide 1.** Make your relations in general go in vertical and horizontal directions only.

Keeping your relations like that immediately makes the view easier to look at. A lot of lines under a variety of angles overwhelms the HVS (Human Visual System). Very rarely, I do use a line segment under an angle. Only when I think it makes the view easier on the eye, I will do it. The same goes for you: develop and follow your 'designer' instincts (which I think you should have if you're an architect). Generally, horizontal/vertical lines work

best.

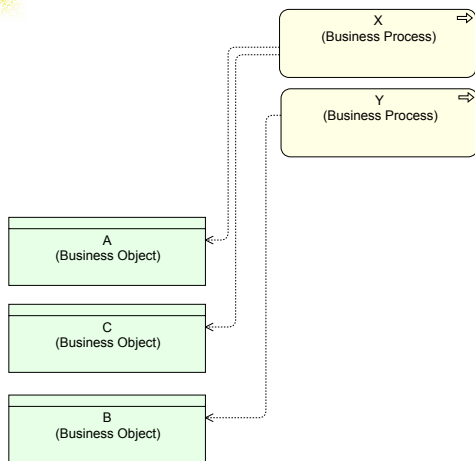
You must watch out for overdoing it, though. Many people will make their view even easier on the eye by using overlapping lines. But overlapping lines have a danger of being ambiguous. Have a look at View 76.

Can you see which Business Objects are accessed by which Business Processes? You can't, so it might be easier on the eye, but it hides essen-



View 76. Style: Overlapping Lines

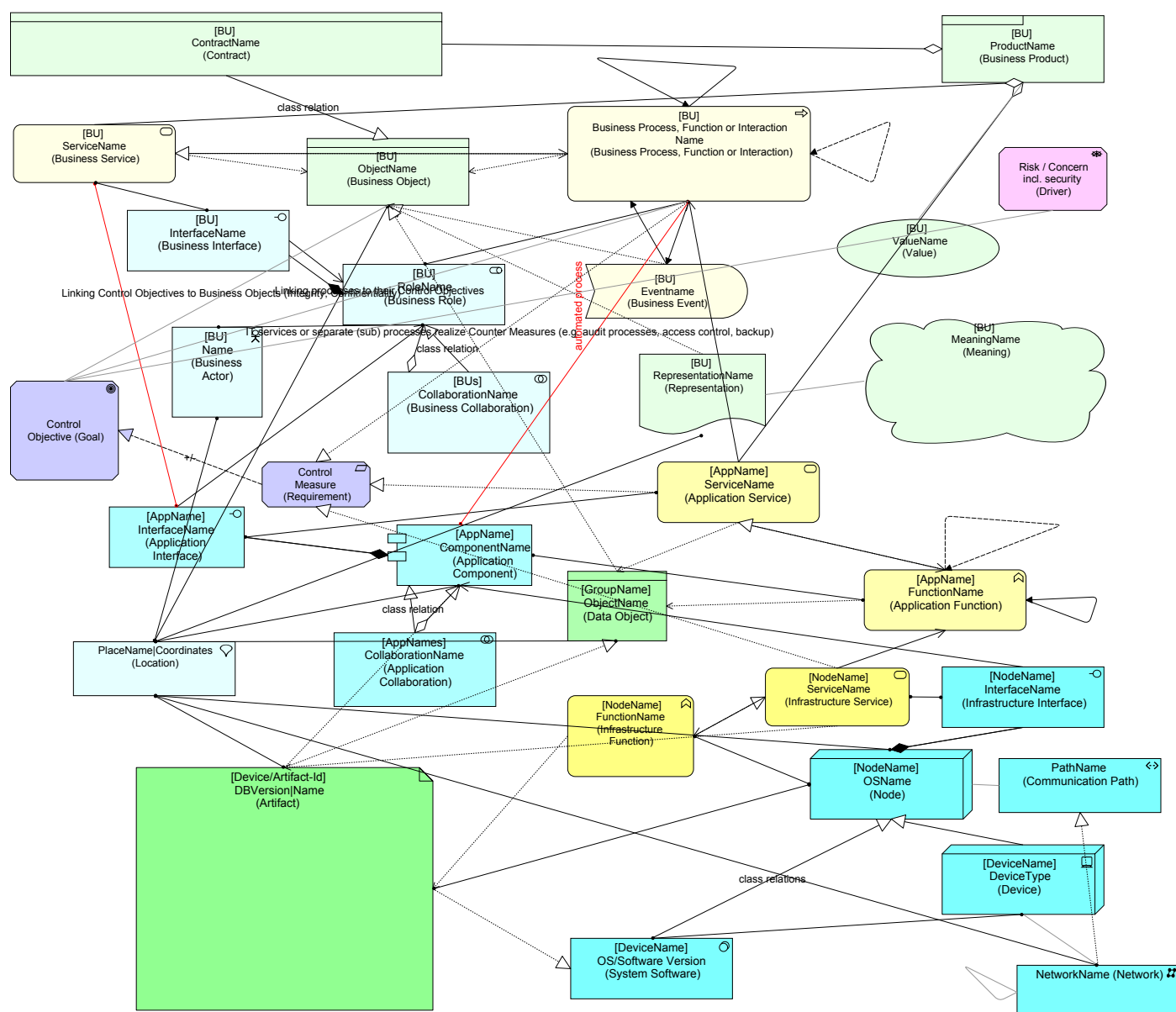
tial information. Therefore, it is generally better to model non-overlapping lines as in View 48



View 48. Style: non-overlapping lines

So the next guideline is:

Style Guide 2. Don't let relations overlap



View 77. ArchiMate Metamodel in Bad Layout

As you can see in View 48, I have also rearranged the order of the Business Objects to prevent lines from crossing each other. Thus:

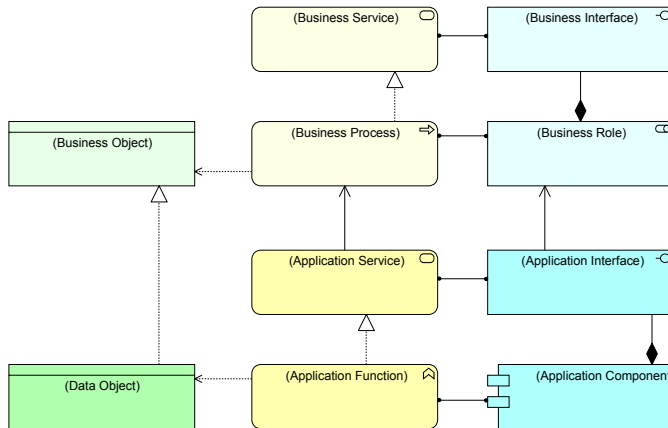
Style Guide 3. Minimize the number of line crossings

And finally, in View 48, I have arranged both Access relations starting from Business Process X to lie close to each other, while having a larger distance to the Access relation starting from Business Process Y. I have created some sort of grouping on the first two relations, supporting the fact that they both are attached to Business Process X. Thus:

Style Guide 4. As much as possible: Group relations according to either source or destination

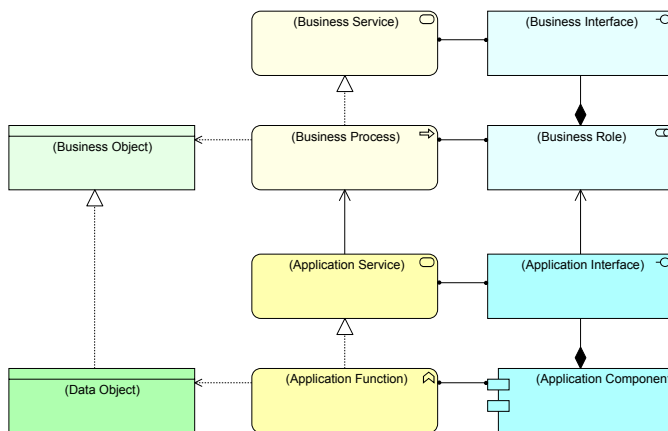
What also helps is to align relations. Say, you have three objects in a row and two relations (between left and midde, and middle and right), it is easier on the eye the have

the relations aligned vertically. Have a look at the following example in View 49:



**View 49.** *Style: Non-aligned Relations*

Now compare with the one in View 50:



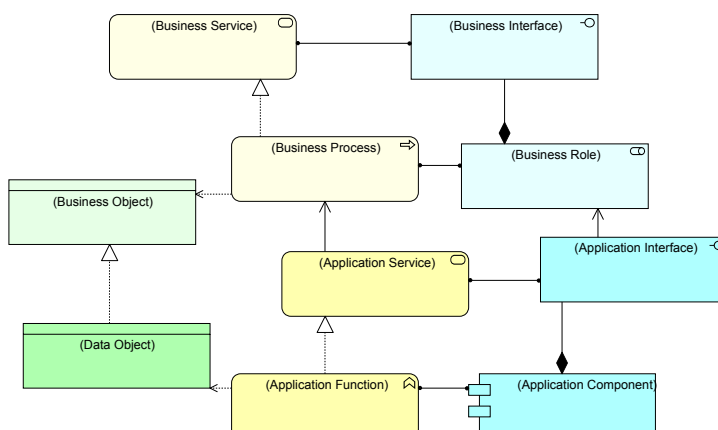
**View 50.** *Style: Aligned and Equally Sized objects, Aligned Relations*

You can see that even in this small view the second one is slightly 'quieter'. The effect is larger the more complex your model becomes. Hence:

**Style Guide 5.** Align relations, even unrelated ones

## 6.3 Sizing and Arranging objects

There are a few simple rules on sizing and arranging your objects that makes the view easier on the eye. So far, the last views above had all the objects created at the same size and laid out in a grid. Have a look at the variation on View 49 in View 51:

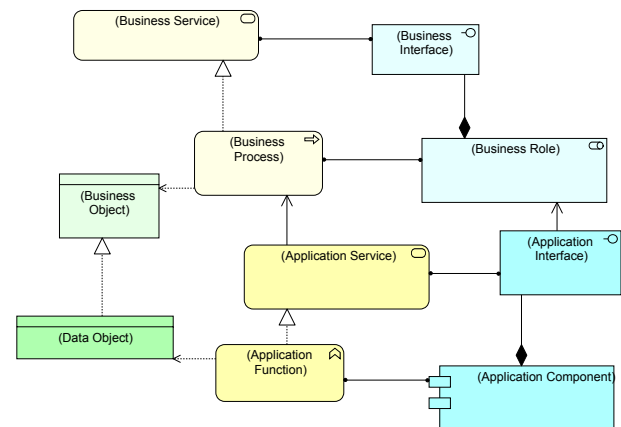


**View 51.** *Style: Equally-sized, non-aligned objects*

As you can see, removing alignment of the objects makes the view again more cluttered. So, the fifth guideline is:

**Style Guide 6.** Align objects, even unrelated ones.

Then, of course, we can make View 51 more noisy even, by varying object sizes as in View 52.



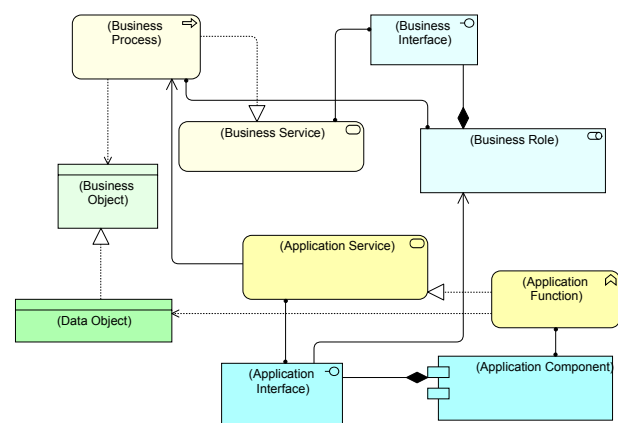
**View 52.** *Style: Variable-sized, non-aligned objects*

This is again more cluttered, so the next guideline is:

**Style Guide 7.** Use as few as possible different object sizes (this is like not using too many font sizes in a text document).

Preferably, most of your objects have the same size. Note: you could have different sizes for different types of objects (say, all application services have one size and all application functions have another), but though the extra distinction might make it easier to spot different object types quickly, having multiple sizes quickly makes more chaos and that chaos overwhelms the advantage of the extra clue to separate object types. Besides, there is a better way to help separate object types (see below). So, in general, use as few sizes as possible in your view.

We can matters even worse still as can be seen in View 53:



**View 53.** *Style: Variable-sized, non-aligned and non-ordered objects*

Just by rearranging the objects, I have created a view with 5 unnecessary line crossings. Note: there are no unnecessary crossings here (under the constraint that I not go around the outside). Try to imagine what would happen if I would change the order of attachments of rela-



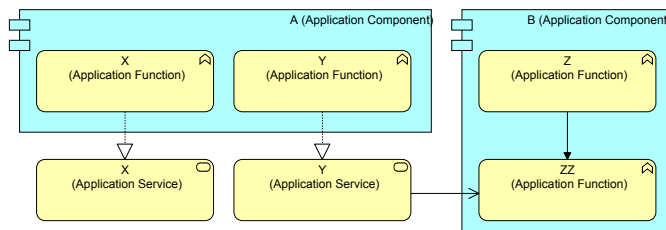
tions on the Application Interface in this example. Hence, the next guidelines are:

**Style Guide 8.** Align objects and attach relations such that relations are as simple as possible and with the least number of line crossings, preferably straight lines from one object to another.

And:

**Style Guide 9.** If you have a nested object or groupings, align the objects that are on the inside as well.

Note that in the example in View 54 Application Function Z is aligned with Application Functions X and Y and that Application Components A and B are also aligned:

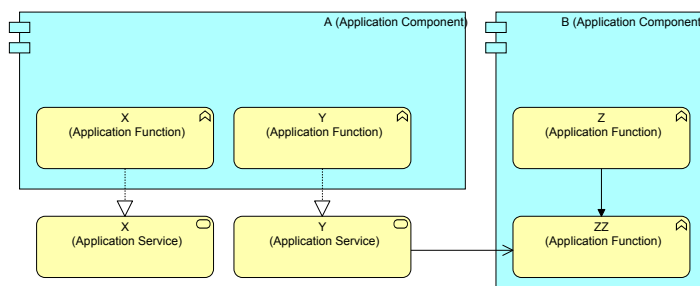


View 54. Style: Align nested objects

Thus:

**Style Guide 10.** Distribute objects evenly within their 'group'.

With a group, I mean that you might have a couple of related objects, I do not mean just actual ArchiMate groupings or nestings, though there it holds too. Such a group in my views will have objects of all the same size, be aligned and evenly distributed. A non-even distributed variation on the previous example can be seen in View 55:



View 55. Style: Objects not distributed evenly in their 'group'

Again, these rules are basic rules. Always, your main goal must be:

**Style Guide 11.** Make a view as easy on the eye, as 'quiet' as possible without losing essential information.

Sometimes, you must break a rule to get the quietness you desire, and so be it, unless you will lose information in the process. Aesthetics is never a matter of just following rules anyway.

## 6.4 Using color

Color is a powerful tool for communication. And it's easy to start using different colors for different (type of) objects in your view, say, you could color all objects of one business unit blue and of another business unit green. I think most of those uses for color can be done with labels and grouping (see below). I use color to mark certain types of

objects consistently to improve readability of the structure I have created.

The official ArchiMate specification is colorless. The original ArchiMate project of Telin (Telematica Instituut), before the first Open Group standardization, used colors in a consistent way, though they were never a part of the original specification, they were a more something of a custom.

Those original 'ArchiMate Colors' were blue, green and yellow. Blue was used for the active objects, like Actor and Role at the business level, Application Component on the application level and Device on the infrastructure level. Yellow was used for behavioral objects, like Business Process and Business Service on the business level, Application Function and Application Service on the application level and Infrastructure Service on the infrastructure level. Green, finally, was used for passive objects, like Business Object at the business level, Data Object at the application level and Artifact on the infrastructure level.

But another way of separation, using the same colors was later introduced as well. This was mainly because there is one 'problem' with the original color scheme, and that is that the 'interface', 'service' and 'data/business' objects all have the same form/icon and would become indistinguishable, and the same is true for 'function' objects. Hence, if such an object was to be found in a view, it would neither from the form, nor from the color be immediately clear if we were looking at a Business Service, an Application Service or an Infrastructure Service. So, some people started to use the yellow/blue/green separation for business layer, application layer and infrastructure layer instead. This way of using colors also became the standard way for some tools.

ArchiMate is color-agnostic, but I have a strong preference for the original coloring which is also still used in the foundational images of the specification. The reason is that one of the strong points of ArchiMate is the way it focuses on behavior as the link between 'actors' and 'acted-upons'. In my modeling work, behavior is actually leading. For instance, when somebody started a discussion on adding the 'owner' of a system to our model, I realized that owner is a role and hence the ArchiMate-driven question arises: what is the behavior (process/function) of this role? And what does this behavior actually change, what is the 'acted-upon' here? This led to what has been described in Section 12 "Secondary and Tertiary Architecture" on page 84.

Using the original color scheme strengthens that basic strong point of ArchiMate because it keeps a focus on the different roles the different objects in the ArchiMate grammar play and I find it especially useful because it supports my preference for modeling with behavior as the central axis around which everything revolves.

Still, it is still handy to be able to discern in a view which layers objects belong to. So, I came up with the color scheme I am using in this book too and which can be seen in View 45 "ArchiMate 2.0 Metamodel" on page 28 (never mind the color used to signal aspects of certain relations in this image, this is just about the colors of the objects)

As you can see there, I kept the blue-for-actors, yellow-for-behavior, green-for-acted-upon separation of the original ArchiMate coloring, but with a twist. I strengthened the colors for the infrastructure level, and softened those of the business level as the higher you go in the meta-model, the 'softer' reality becomes. Colleague Joost Melsen tells me:

*Our usual demarcation is such, that a business process allows for a larger degree of freedom in behavior than an application function, which in turn is 'richer' in behavior than an infrastructure function. We will probably never take away too much freedom on the human processing level in favor of IT, because it allows each of us humans to act and decide according to our own values of correctness, and thus be meaningful*

Now, if we create overview views with all levels represented (e.g. for a Project Start Architecture), it is with this color scheme immediately clear what is business, what is application and what is infrastructure. It is also immediately clear what are actors, what is their behavior and what does this behavior change. It is easy to explain this to business and developers alike.

## 6.5 Grouping

ArchiMate has an official 'grouping' relation (see Section 2.8 "The Grouping Relation" on page 26) and for three relations (Composition, Aggregation and Assignment), you are also allowed to depict it by Nesting (see Section 1.8 "Nesting" on page 20). Here, I want to say something about 'visual grouping' as a means to keep views easy on the eye. For this, you can of course use the ArchiMate Grouping relation, but you can also do it purely visual. For instance, you can put all the objects from the Business Layer close together, so that you can focus on one part of the view to see the business aspects. Or you can group on the basis of subprocesses in a view. The goal here is to make the layout of your view such that they eyes have to travel a minimum distance, based on what you want to convey. We'll see a few uses later in a few examples in the book.

## 6.6 About labels

ArchiMate has been labeled an 'architecture language'. It certainly has aspects of a language. The combination of actors, behaviors and acted-upon objects resembles subject-verb-object of a normal language. And that is intentional: the designers of the language had this in mind. But what makes a language?

There are two issues I'd like to discuss here (and yes, they have a practical application in Enterprise Architecture work):

- Redundancy in a language
- Syntax versus semantics (grammar versus meaning)

To start with the latter: take the following natural language sentence: "The customer drinks a bicycle". Assuming that there is no cocktail called 'the bicycle', this is not a meaningful sentence in the English language. You can't *drink*

a bicycle. Grammatically (syntactically), the sentence is correct, but semantically, it is nonsense.

We can do the same in ArchiMate. It is easy to use relations to connect objects in a grammatically correct but meaningless way. E.g. using a word processing application to steer the magnets of the Large Hadron Collider in Geneva. Easy to model in ArchiMate, but meaningless in practice.

So, not only the *form* your sentences take is important, but also the *words* themselves. So, take a good look at ArchiMate: its objects are word-types, *not* actual words. Without a label in each object, views become meaningless (incidentally, we use an anonymizing script in our tool before we send models to the tool company's help desk to help prevent the leaking of confidential business information). ArchiMate, therefore, looks more like an architecture-*grammar*, than an architecture-*language*. This means that thinking about the labels becomes important. If you are language speaker with perfect control of grammar, you still need to find the right words.

The single most effective choice I made with regard to communicating our views was to keep the *type name* of an object in the *label* of an object. This is of course redundant, as (with the right color coding) every object type is already visually different from another. But it is very effective. For people who do not read models every day, the addition '(Business Process)' at the end of the label hugely speeds up understanding the model. 'Business Process' immediately triggers the right concept in their mind, the icon itself does not. This is even true for experienced architects that use ArchiMate every day. In fact, adding this simple redundancy has in my opinion immensely enhanced the usability of ArchiMate views in our organization. For instance, already early on, infrastructure specialists expressed an interest to replace their own visualizations (spreadsheets, drawings) with our ArchiMate views. That is something you generally are told is not to be expected. "Don't expect to be able to use large and complex ArchiMate views outside of your core architect group", you are told when you go to your first ArchiMate course. In practice: a little redundancy goes a long way towards readability.

I also use other guidelines for labeling of objects. These have to do with easy use and especially reconciliation and integration with other models that exist in other tools in the organization, e.g. the CMDB, the process- or risk modeling tool, the software design tool and others.

So, is this practical? In my experience yes: forget the nice grammar for a moment: the power of language in the end lies in semantics. And there, redundancy improves communicability. There is nothing new there: nature has done it first: natural language is full of communication-enhancing redundancy (to the great pleasure of code breakers through the ages, by the way).

Architects are normally inclined to simplify, have a single definition, etc., in other words do everything that reduces complexity, to rationalize — to make logical. But Enterprise Architecture relates strongly to the real world, where logic, certainly pure logic, has its limitations and much of it actually may make matters worse.

The pattern I use generally puts three things in a label:

- First line: some grouping information between square brackets. In the infrastructure layer, generally the device name, e.g. '[winsrv001]', in the application layer, generally the application name and in the business layer the business unit name.
- Last line: type name between brackets, e.g. '(Application Service)' for the Application Service.
- In between: whatever short labeling you want there, but generally the 'name' of the object as ArchiMate calls it.

## 7. Patterns

*Note: From now on, it might be practical to keep a finger on page 131 so you can easily move there and back.*

### 7.1 The Use of Patterns

Asking "How do you model X?" is probably the most asked question by those starting to use ArchiMate. This chapter intends to offer solutions for many such questions, like "How do you model a spreadsheet?" or "How do you model a Business Rule Engine?". These patterns are certainly not the only way to do it and I might even offer alternatives. In the end, if you become experienced, you will tend to create your own patterns. Modeling style is after all like writing style. And good modeling is like good writing: a mixture of good content and good style.

But there is more to patterns than just a nice way to model something. If you end up using ArchiMate to model your Current State Architecture, the model that contains your existing Enterprise Architecture, that model becomes much more usable (and thus following Uncle Ludwig, much more meaningful) because using the same pattern in comparable situations for instance enables you to create (depending on your tool of choice) automated exports and analyses.

Imagine you have that large Current State model and it enables you to do an analysis that can tell you that if a certain Business Process needs a Return To Operations (RTO) of 4 hours, you can easily find all the servers that need to support that requirement. Or if you have a Information Manager, you can easily find which applications he or she is responsible for. Or you have a server nearing the point where its capabilities are overwhelmed and you can easily identify the business processes responsible. ArchiMate is an Enterprise Architecture language, which entails *all* the layers and it is well suited to provide the foundation for such information.

In the coming sections I will present several patterns. I will start with a basic Infrastructural pattern. The reason for that is that this pattern will return in a few guises and also as underlying pattern for some application patterns. So, without further ado, here is:

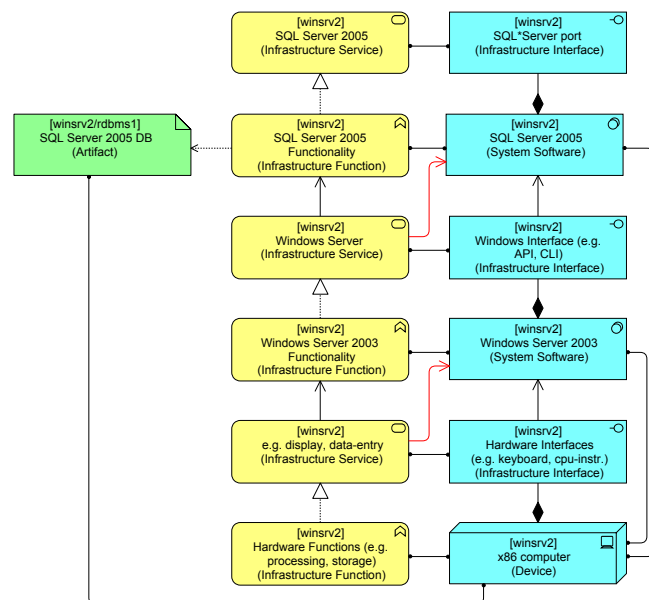
### 6.7 Don't Use the 'Children's Forms'

After having gone into things you can do to improve readability, here is in my view one thing you generally should *not* do: use ArchiMate's 'playful' object forms like the big puppet for Actor or the big Powerpoint-like arrow for Business Process, etc.. In my opinion, these forms are for children, not for grown ups and they are not visually 'quiet'. Maybe for the absolutely simplest views with only a few objects that are shown to people who never have to look at ArchiMate views again, but in that case I would probably not use ArchiMate at all but create a funny drawing with Greek columns, jumping dogs, images of police constables, etc.. ArchiMate is intended for serious messages and using the 'funny forms' is akin to writing your serious design document in limerick form. Your views are serious business, they should look the part.

### 7.2 A Basic TI Pattern: A Database

ArchiMate has its three layers and some of what happens in your architecture will be classified as Infrastructure. In ArchiMate, that is more than just hardware. It is also artifacts like files and it is software with its behavior: operating systems, surely, but also other systems like relational databases and so forth. ArchiMate has software on two levels, and though it is useful it can also be confusing. Because if you have software, how do you model it? As System Software or as Application Component? If you look at it from an IT perspective, the difference is illogical: after all software is software. But from a business perspective it makes kind of sense. For the business, its business-critical applications are what count as 'application', the software beneath it like databases are 'infrastructure'.

If you want a fully detailed model for a database at the infrastructure level, you get something like View 56:

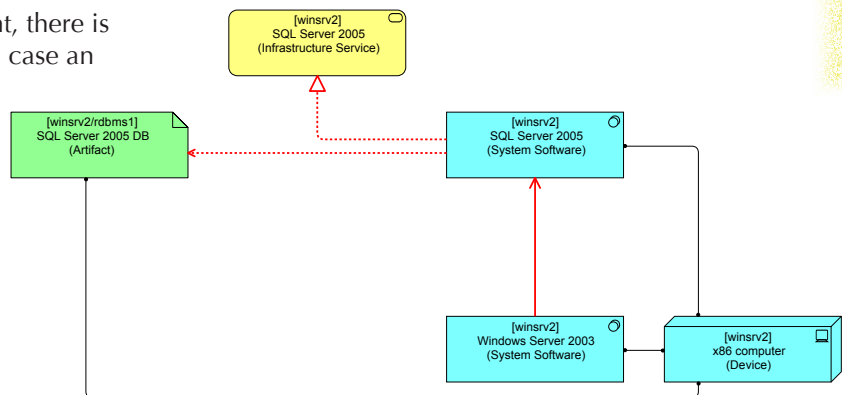


View 56. Full details of deploying a database

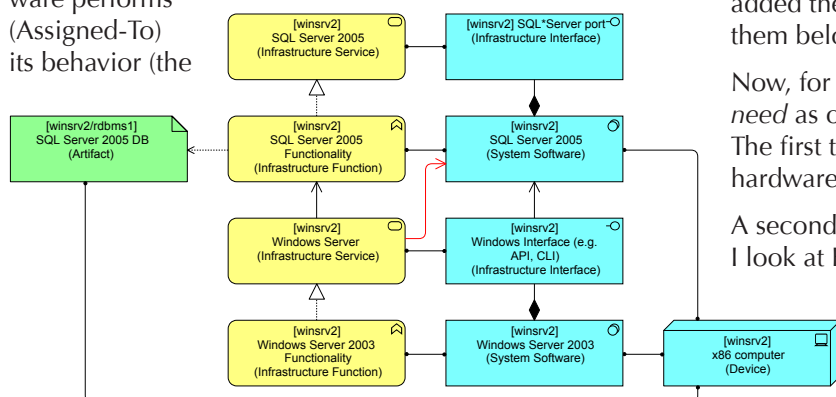
Unless you are an Infrastructure Architect interested in modeling these details, you will probably never model it like this. But it is the basis from which we are going to simplify to get to the practical pattern we will use.



This is what is modeled in View 56: Bottom-right, there is a Device, the actual computer hardware, in this case an x86 server. This is hardware, thus an *active* object, and thus it has *behavior*: it has functions it performs and services it provides. Installed on the Device (the Device is Assigned-To the System Software) is an Operating System. This active object again has behavior: the functionality of the Operating System and the services it can provide. Its functionality requires the use of the hardware services that the *Device's* functionality provides. Also installed on the Device (Assigned-To it) is the SQL\*Server 2005 Relational Database Management System (RDBMS). The SQL\*Server 2005 System Software performs (Assigned-To) its behavior (the



**View 80.** Deploying a database, without hardware details, interfaces and internal behavior



**View 78.** Deploying a database, without the hardware details

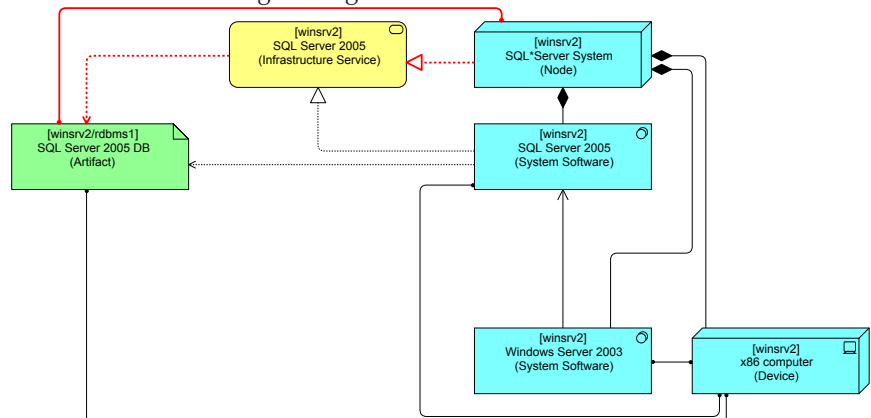
SQL\*Server 2005 Functionality Infrastructure Function) which in turn Realizes a SQL Server 2005 Infrastructure Service which can be used by Application Functions elsewhere in your model. The SQL\*Server 2005 Functionality requires the Operating System's Infrastructure Services to function. On the left, there is the actual database Artifact, the Device is Assigned-To this Artifact, signaling the Artifact resides on the host. The SQL\*Server's functionality accesses that database artifact.

Since the Assigned-To relation in ArchiMate 2.0 is unidirectional (from active object to behavior), ArchiMate 2.0 has a few explicit extra relations to show that all active objects can use services. Normally, with such a detailed view, we would not add these. I have

added these in red in View 56, though, because we use them below to simplify.

Now, for the pattern I want to use, I must decide what I need as objects for future use, e.g. Analysis or reporting. The first thing I really do not need is all the details of the hardware. Removing those gives me View 78.

A second thing I generally do not model are interfaces. I look at Enterprise Architecture from the perspective of *behavior*. I am interested how behavior of the applications are used by the business for instance. I am interested in the services the infrastructure must provide for the applications to function. How this is technically done (Infrastructure Services for instance represent infrastructure protocols like HTTP, SMTP or database server protocols) is not something that interests me as an Enterprise Architect, though it might of course interest the TI-Architect.

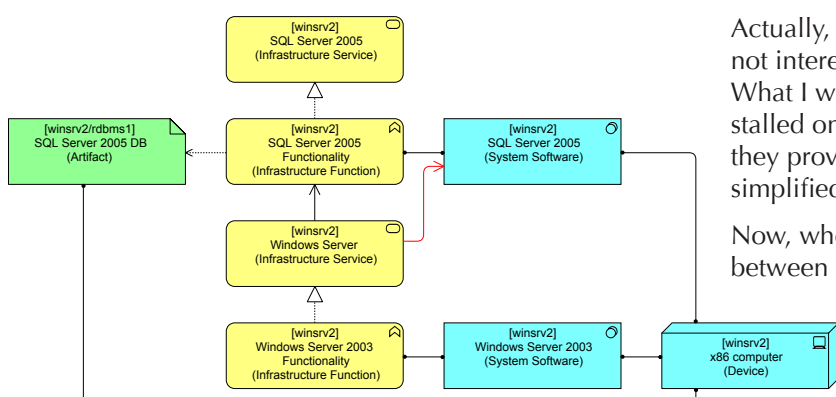


**View 81.** Deploying a database: using a Node to make an abstract device

Without Interfaces, View 78 turns into View 79.

Actually, for reasons of Enterprise Architecture, I am also not interested in the internal behavior of my infrastructure. What I want to know what System Software has been installed on it and what Infrastructure Services and Artifacts they provide for the outside world. So my View 79 can be simplified to View 80.

Now, when I removed the internal behavior objects, a link between active infrastructure objects and the Infrastructure Service and Artifact that are exposed to the outside world was lost. These have been replaced by derived relations (in red). For instance: from the Windows Server System Software via Assigned-to the Windows Serv-

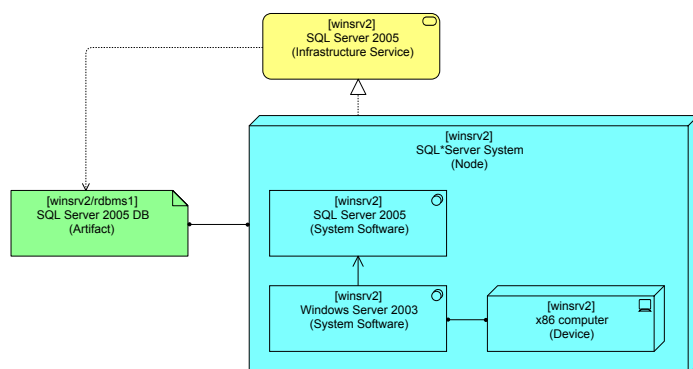


**View 79.** Deploying a database, without hardware details and interfaces

er Functionality via Realization to the Windows Server Infrastructure Service via Used-by to the SQL\*Server 2005 System Software (here we need that extra Used-by from View 79).

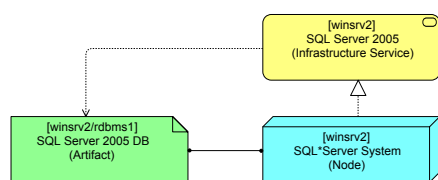
For infrastructure, I can adopt the use of an intermediate (abstract) Node to encapsulate everything below what is exposed to the outside world as already suggested in Section 1.9 “Using a Node to encapsulate infrastructure” on page 21. This is shown in View 81 on page 45. This Node has the internal System Software and Device objects as Composite parts. From the Node, we can create new derived relations from that Node to the Artifact and Infrastructure Service that is exposed to the outside world. These are shown in red in View 81.

When we then use Nesting to get all the Composite parts inside the Node, we get View 57:



**View 57.** Deploying a database: Nesting details in a Node

This is actually the pattern I prefer for modeling basic infrastructure items. I do not model all the omitted intermediary objects and internal structures that I used to come to this summary pattern. I used derived relations to get this result. There is no need for the other details in my Project Start Architectures or Current State Architecture. But even if I model the above, I can still when it suits me present an



**View 58.** Deploying a database: Collapsing the Node

even simpler view on that pattern as can be seen in View 58.

Now, this is a pretty simple

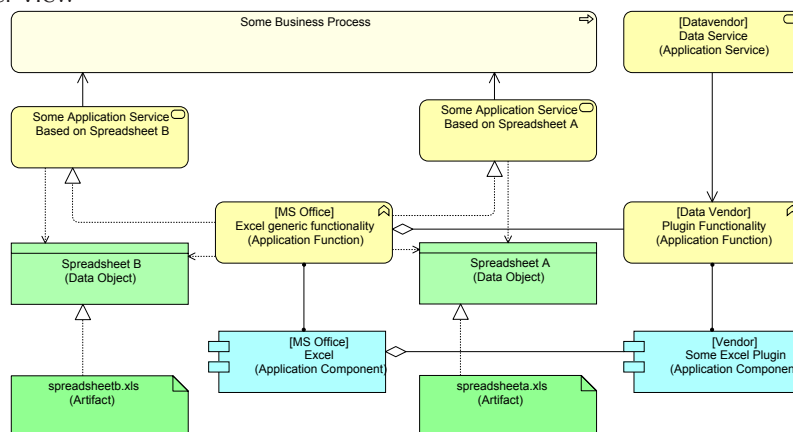
TI-structure pattern. We have a Node that embeds a single Device to which two System Software objects are Assigned. We Realize the Infrastructure Service directly from the Node, making the Node (and the Infrastructure Service and the Artifact) a sort of cut-off between Infrastructure use and Infrastructure structure. The Node is used in fact as a sort of object in an object-oriented approach, hiding its internals, exposing its interfaces. Later, we will see a few more complex examples when we look at ‘clustered’ infrastructure setups. An important aside: a server can of course Realize multiple Infrastructure Services, say a RDBMS service and a File Share.

## 7.3 Modeling Spreadsheet Applications

Excel Spreadsheets are IT’s (and an Enterprise Architect’s) nightmare. First, there is no way of tracking them, so you may have all kind of mission critical stuff in spreadsheets that have no access control, no logging and auditing, and which are written poorly, etc.. But except being a nightmare, they are also a fact of life in any modern organization. And it is worse: Excel is often augmented by all kinds of plug-ins from all kinds of vendors. SaaS-vendors, Data vendors, functionality vendors etc. distribute plug-ins (statistical plug-ins, access to tooling plug-ins, etc.) that become part of the Excel application environment.

So, all that mission critical application behavior that is in Excel is something that gives headaches to Enterprise Architects (and Support Organizations) because it generally is not tractable. In architecture descriptions, people end up with models that have objects like ‘Excel’ in them that stand for ‘anything that is done in Excel’ and that generally means ‘we have no idea at all what is happening there’.

Personally, I think most of what is in Excel is not a real problem (except for that access control, auditing and logging and such, if these are required) for Enterprise Architects if looked at properly. First, those little spreadsheets that employees create for their daily job are mostly not that mission critical at all. They tend to improve the efficiency of what they otherwise would have to do by hand (and can do by hand if need be). (Some, however, are not that innocent at all, like the spreadsheet I once saw that had all the information of another application copied into it for easy reference and that was two years out of date). Also, in some organizations you will find literally hundreds of thousands of spreadsheets, that on closer inspection tend to be some sort of smart form. One person enters data, another reads the data. And afterwards the form is never used again. It sits on your servers as an archive copy of something that happened once. It’s not an active object, it is an old passive object, which probably



**View 82.** Excel as an Application Component

never will bother you again.

But, if you look into your organization, you will probably find there are indeed some mission critical applications that are in fact spreadsheets and for which there is no business case to turn them into properly built and maintained applications. Think of the analysis models Asset Management Portfolio Managers use and play around with. Turning these in formal applications with change procedures and lead times of weeks if not months defeats

their object. You still want these in your models, though, and you don't want them in there as a generic 'Excel' object. So, there is need to decide on how to model a spreadsheet application.

Initially, you might want to start with adding Excel as an Application Component to your landscape as shown in View 82.

Excel's functionality Accesses 'spreadsheet' Data Objects that are Realized by .xls Artifacts. There are two spreadsheets in this example, one of which depends on a vendor plug-in that connects to an outside data service. Think for instance a stock market plug-in that delivers live price data for stocks and bonds.

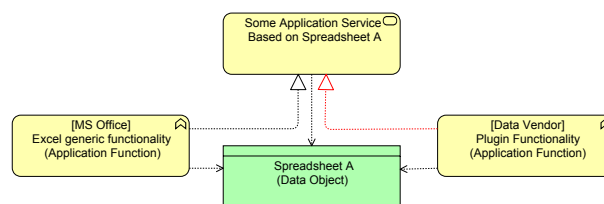
In the example, spreadsheet B is a normal spreadsheet based on Excel's functionality only. Its data is contained in the spreadsheet and so is its logic (macros, programming). The Excel Application Component is Assigned-To the generic Excel Application Function (all of Excel's functionality). This functionality reads the spreadsheet data object, and with the logic contained in the Data Object and based on its internal engine to process that logic, it Realizes the Application Service that is used in a Business Process.

Spreadsheet A requires the use of a vendor plug-in. This is modeled as follows here: the plug-in is generally installed as a part of the Excel installation. On the infrastructure level we are generally talking of some sort of Windows dynamic loadable library (DLL) which is loaded by Excel and the plug-in becomes a component that becomes part of Excel. This has been modeled with an Aggregation. On the functional side, both Application Components have their Application Function, covering everything what the Components can do. The main Excel Application Function is in the lead and it Accesses spreadsheet A, and realizes Application Service A. It is the plugin's functionality that accesses the outside vendor's Application or Infrastructure Service.

So far so good, but there are a few disadvantages to modeling it this way:

- Because the generic Excel functionality is in charge of Realizing all Application Services, everything is related via that Application Function. Application Service B depends on the plug-in, while it does not need the plug-in. It is in fact not possible to discriminate between Application Service A and B if you want to know who depends on the vendor's data service and plug-in.
- The .xls artifact contains both data and logic, but the logic does not end up in a recognizable Application Function. Application Functions are nice to have when discussing creation and maintenance of IT.

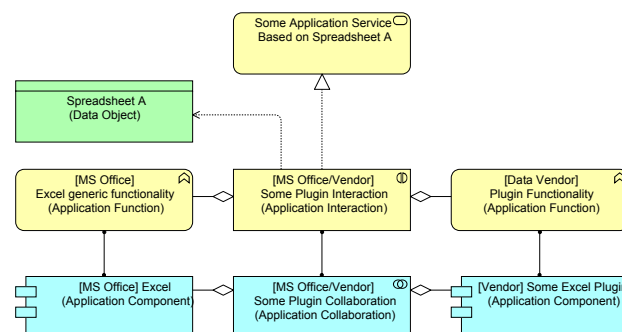
The first disadvantage can be solved by having a relation between the plugin's function and Application Service A (and not B) as seen in View 59.



**View 59.** Excel as an Application: linking a plugin to the Application Service

Though not wrong in the sense of the language, the Realization (in red) is not true in a real sense of course, because it is Excel that creates the Application Service *using* the plug-in, and not the plug-in directly in some way.

If you want to do this properly in ArchiMate, you should in fact use Application Collaboration and an Application Interaction as in View 57.



**View 60.** Excel and the plugin as an Application Collaboration

Both alternatives are not really attractive to me, so on the application level I would stay with the first solution. And the dependence of spreadsheet B on the plug-in is not really a problem: after all, the functionality of the plug-in could actually still break spreadsheet B, so in that sense the dependency is still real as we noticed in 5.4 "On the direction of structural relations under the assumption of 'worst case'" on page 34.

The other disadvantage (no visibility of the functionality, the behavior, that comes from the logic (macros and such) inside the spreadsheet) remains. A spreadsheet itself can be a kind of application, people 'program' spreadsheets, but this application nowhere shows up as such in this approach. That is why I prefer another solution. I consider the .xls as an Artifact that Realizes *both* a (passive) Data Object and an (active) Application Component. The spreadsheet itself is an application, not just data. Excel then becomes the (generic) *platform* on which the application runs, a bit like a Java Virtual Machine being the platform within which the Java application (that is Realized by a '.jar' Artifact) runs. This can be seen in View 83 on page 48.

Here, the spreadsheet.xls Artifact Realizes not only a Data Object from the data in the spreadsheet, but it also Realizes an Application Component from the logic in the spreadsheet. That Application Component is assigned to the Application Function that stands for the *behavior* of the spreadsheet. This behavior has access to the data (in fact, the Excel formulas and Macros have access to the (passive)



data that is *also* in the spreadsheet, this happens internally). The behavior then Realizes the Application Service for B that is used by the Business Process. This can be extended to a spreadsheet like A that uses the plug-in to access a vendor's data service. This can be modeled in various ways, here it is modeled as the plug-in being an aggregate child of the Excel Infrastructure Service so that we still have the link between the plug-in and Excel (if the plug-in breaks, Excel can be affected). But for the rest, the plug-in service is used by the Application Function that represents the behavior of spreadsheet A. Also modeled is a direct Used-By at both the application layer and the infrastructure layer to show the use of the vendor's data service. Technically, the Used-By from the data vendor's Infrastructure Service to the plugin's Infrastructure Service is not fully modeled. The data vendor's Infrastructure Service should ArchiMate-technically be modeled as being Used-By an Infrastructure Function, which in its turn Realizes the plugin's Infrastructure Service. As it is, the Used-By is a derived relation.

The biggest advantage of modeling Excel as an Infrastructure Service is not that stuff with the plug-ins. It is that it so clearly shows that a spreadsheet can be an *application*. Spreadsheets are used as passive objects and as active objects. We tend to think of passive objects as 'harmless' but the fact that all those spreadsheets are the nightmare for IT and Enterprise Architects is because they often are not harmless passive objects at all, they often are applications and should be treated that way. And that Excel wizard Asset Manager is a programmer even if he or she does not think so.

Next in line: another 'application' that is no application.

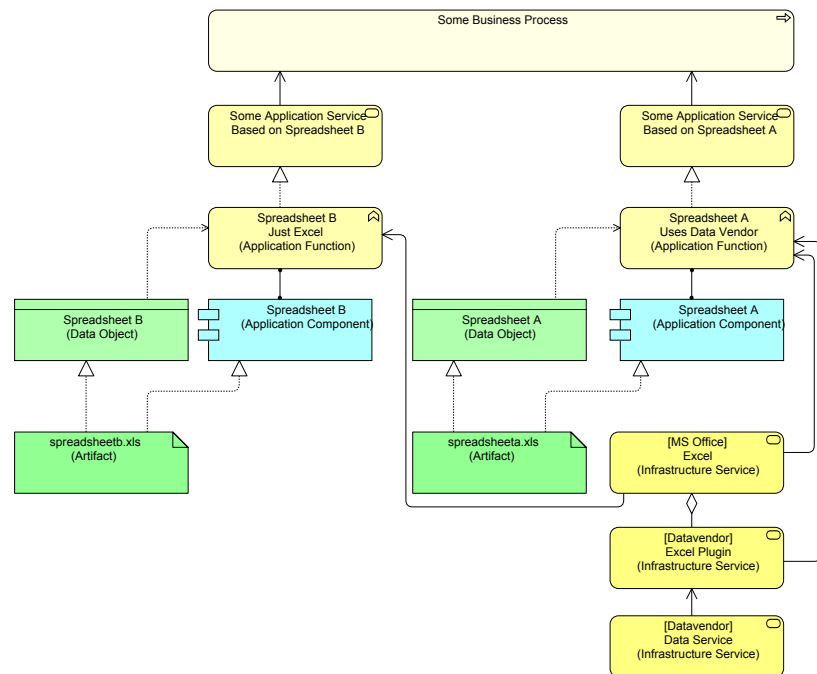
## 7.4 Modeling an Internet Browser

It is illustrative to think about the reality of what happens when you use a web site. You start up your browser and you type in an URL, e.g.:

`https://archimatemusings.wordpress.com/wp-admin/post.php?post=99&action=edit&message=10`

The browser connects to the web server and loads a web page that the web server sends. What the web server sends is based on the URL, which is composed of three parts: a protocol identifier (e.g. `https://`), a domain name (e.g. `archimatemusings.wordpress.com`) and the request part (the rest of the URL, e.g. `/wp-admin/post.php?post=99&action=edit&message=10`).

Now, based on that request part, the web server returns bytes to the browser using the 'https' protocol as requested. That data could theoretically be anything, but in this case, the data conforms to a standard language, HTML, which the browser understands. HTML originally was just a simple description of (passive) markup/layout, but browsers these days are far more powerful. They have functionality that allows them to present *active* components to the user, say buttons, pop-ups and so forth and ways to run almost arbitrary code. This code is normally JavaScript code, which is an application platform built into almost every browser. JavaScript, by the way, has nothing



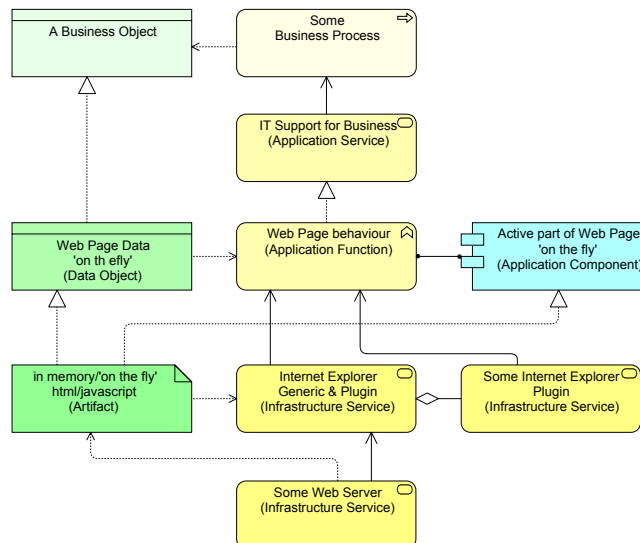
View 83. Excel as Infrastructure Service (Application Platform)

to do with the programming environment 'Java'. JavaScript was originally called "Mocha", then "LiveScript" until somebody at Netscape thought it a good idea to change the name so it could profit from the hype surrounding Java at that time. But I digress.

Anyway, what is being sent from web server to web browser is just a file, a stream of bytes. In ArchiMate terms, that is an *Artifact*. Now, this Artifact Realizes both passive (the text you read) and active (behavioral) elements like those buttons, clickable maps and so forth. What actually happens is that your browser interprets the file (html/JavaScript) and realizes *both* a passive Data Object and an active Application Component. This is just like Excel did in the previous pattern: Excel interprets the spreadsheet and Realizes both passive (Data Object) and active (Application Component) objects.

But there is a difference. With a web browser getting the file from a web server, the file may be just 'in memory' and the whole setup is created 'on the fly'. There is no Artifact on a disk anywhere, it may just exist fleetingly in

memory of the desktop running the browser. Close the browser and it is gone. The pattern is shown in View 61



**View 61.** The Internet Browser as Infrastructure Service (Application Platform)

This pattern is useful to make clear where the application is. Sometimes you see models where the web site is modeled as an application. But that muddles the picture because it suggests that the application does not run on your infrastructure. In reality, with browser-based applications, the application runs *in the browser platform* which in turn is part of your own infrastructure. It *looks* like everything runs elsewhere, but that is not true. It only *seems* that way because it is downloaded 'on-the-fly' and destroyed as soon as you quit the browser. You have a temporary, downloaded application that generally connects back to the same server (or another one) to interact with. The only reason that your security people do not get scared about you downloading and executing applications is because the Javascript run time in the browser is a 'sand box' that is properly separated from your local infrastructure. But change the Javascript run time to some browser plug-in that is used as platform, e.g. Flash, and your security people will become restless. Because Flash for instance is a plug-in that is known for security breaches.

What we have seen with the spreadsheet and web browser so far is that they are not so much applications per se, they represent a *platform*. This idea of looking at platforms is very useful when modeling.

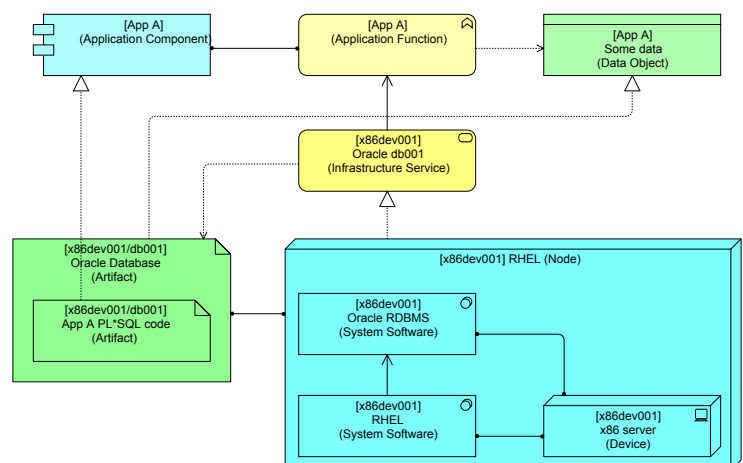
Note: this is not the entire story. After all, what is sent to us is often just the 'presentation layer' and not the entire functionality. We'll get back to this in 7.13 "Deployment Pattern: Three-Tier Application" on page 55.

## 7.5 More 'application platforms'

So what are good example of application platforms of which we now have seen two (spreadsheet and browser)? Here are a few:

- Javascript (Browser built-in, loads parts of the html page, namely the part inside a `<script type="text/javascript">` tag. Another script language that can be delivered inside an html Artifact is for instance VBScript (only Microsoft browsers support this).

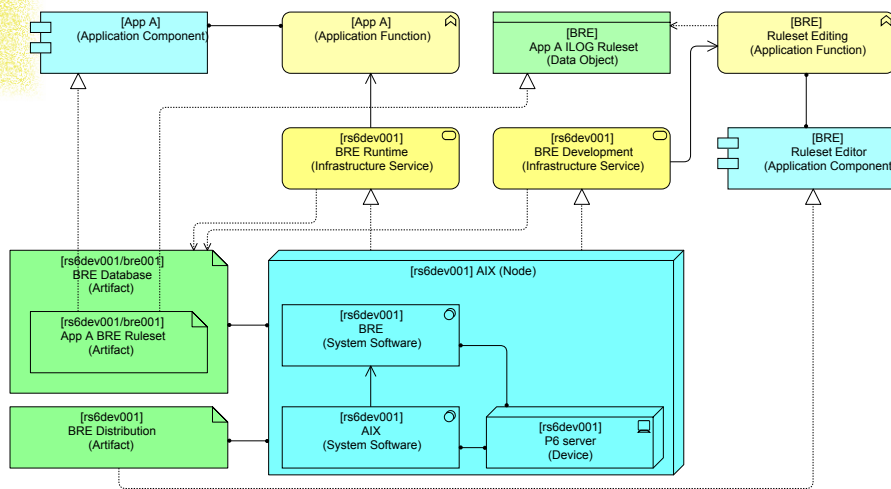
- Tcl is another scripting language that can be delivered inside an html document with a `<script>` tag. It requires a plug-in.
- Adobe Flash (Browser plug-in). Loads .swf ("shock-wave-flash") Artifacts and Realizes Flash applications)
- Microsoft Silverlight (like Flash)
- The above are browser-based platforms. But there are others:
- Java (Java Virtual Machine that loads .jar Artifacts which Realize Java applications). Sometimes the Artifact is also delivered via a browser, but the downloaded Artifact runs in its own Java sandbox, not in the browser.
- Perl, Python, Ruby, etc. are languages where a script is read by an application that acts like a platform. This is like the earlier Excel pattern. Perl is not the application, but the perl script is an Artifact that Realizes an Application Component.
- Unix shell scripts or Windows .bat or .cmd batch files are examples of Artifacts that are read by platforms and turned into Application Components. The platform here is the operating system.
- Enterprise Service Buses.
- Relational Databases. These may have programming languages embedded. E.g. In Oracle you can pro-



**View 84.** Database as an Application Platform

gram in PL\*SQL: you can see how this would look, expanding on the earlier basic infrastructure pattern which was also explained using a database. As you can see in View 84, the PL\*SQL code is an Artifact that is a composite part of the database it resides in. This PL\*SQL code Artifact Realizes an Application Component, which uses the Oracle db001 Infrastructure Service, both to be able to run the logic as well as access the data in the database.

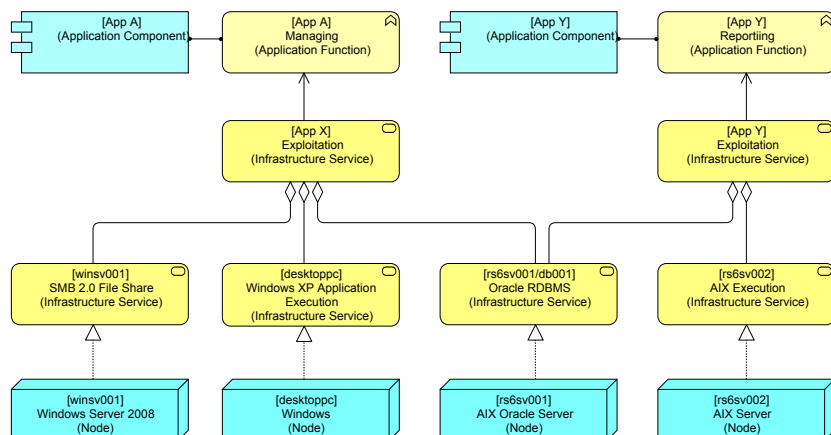
- Business Rule Engines. This one is illustrated in View 85 on page 50. This example is a bit more complete as a platform than the previous ones. Here you see that the rule set is modeled as an Artifact that Realizes an Application Component, which uses the BRE runtime environment to function. Next to the runtime, you see that the BRE distribution also contains a development environment to create and edit the rule sets. In this example the development environment



View 85. BRE as Application Platform

runs on the server. In reality, this probably will be some sort of browser based or Java-based environment that has a thin client somewhere and Artifacts will be promoted from development to a runtime, but for simplicity of this example it has been assumed here that development happens on the runtime server. An interesting aspect of this example is that the rule-set Artifact Realizes *both* an Application Component and a Data Object. For the developer, the rule-set is *data*. But for the business, the rule-set is an *application* that is used in its business process. That business process is primary to the business, the development of the rule-sets is secondary. That is why I call the direct use of IT by the business Primary Architecture and processes like application management or application development Secondary Architecture. There is also Tertiary Architecture (e.g. The processes of Enterprise Architecture, leading to for instance requirements for Primary and Secondary Architecture. We'll get back to that in a section 12 "Secondary and Tertiary Architecture" on page 84.

If you add all these platforms to your landscape, you will begin to appreciate the amount of programming still going on in your environment. This is what I would call 'hidden programming'. And because it is hidden we tend not to see the hidden complexity we have created in our environments. Not only the end users tend to do their 'end user computing' (EUC). These days, looking at EUC is on many organization's radar. But your infrastructure people who write perl and shell scripts, batch jobs and so forth are not on many radar screens yet. They happily go on



View 86. TI Building Blocks

creating their landscape of many little applications without much notice, until someone want to change something, that is. And that simple off the shelf database might hide application programming as well. When we are done looking at EUC, we might pick up IMC (Infrastructure Management Computing). And after that, we might realize that Business Rule Engines do not remove programming, they just move it from the programmer to that poor business analyst who, just like the sales manager programming his Excel spreadsheet, is a programmer even if he of she does not realize it.

Summarizing: if an environment can be programmed and it provides a kind of run-time environment, you probably are looking at something that can be modeled using the 'platform' approach. Modeling the applications hidden in these platforms is necessary if you want a complete view of your Enterprise Architecture, because they contain essential business logic.

## 7.6 Infrastructure 'Building Blocks'

In View 57 on page 46 we encountered a basic TI Pattern. The TI Pattern is that you model the TI as a single Node that hides the internal complexity, then you let that Node Realize Infrastructure Services and the Node is Assigned-To any related Artifact. This makes the Infrastructure Service and the Artifact the linking pins between the infrastructure layer and the application layer.

But in most enterprises, infrastructure is shared between various applications. A file share may be used by many applications, and even databases may be shared by different applications, say, one application that uses the database to store transactions and another separate application that creates management reports on those transactions from the same database. Both use the same database and thus both use the same database Infrastructure Service. The way I prefer to do it is to have the Infrastructure Services clearly named for the Nodes that realize them, whereas an abstract 'aggregate' Infrastructure Service aggregates these into the specific set that is needed for each application to function (and hence is Used-By the Application Function). This abstract Infrastructure Service is then named the 'Exploitation' Infrastructure Service for that particular application, and it is the ideal service for the infrastructure people to provide a Service Level Agreement for running the application. It looks like View 86.

In this example, App A (the 'transaction application') runs on a Windows desktop and uses both a file share and a database. App Y (the 'reporting application') runs on a RS6000 server and accesses the same database as App A.

There is a big advantage to this pattern: the application layer of your architecture generally does not change if the deployment changes.



The pattern has a slight disadvantage: given the direction of the Aggregation relation, you cannot create a derived Used-by relation from either of the Nodes to either of the applications, whereas you know that this relation certainly exist. The abstract 'aggregate' Exploitation Infrastructure Services break the possibility to derive the relation in ArchiMate terms. In reality, this is not really a problem because tooling will generally allow analyses that go beyond derived relations and tooling will allow you to traverse the aggregation relation in the opposite direction.

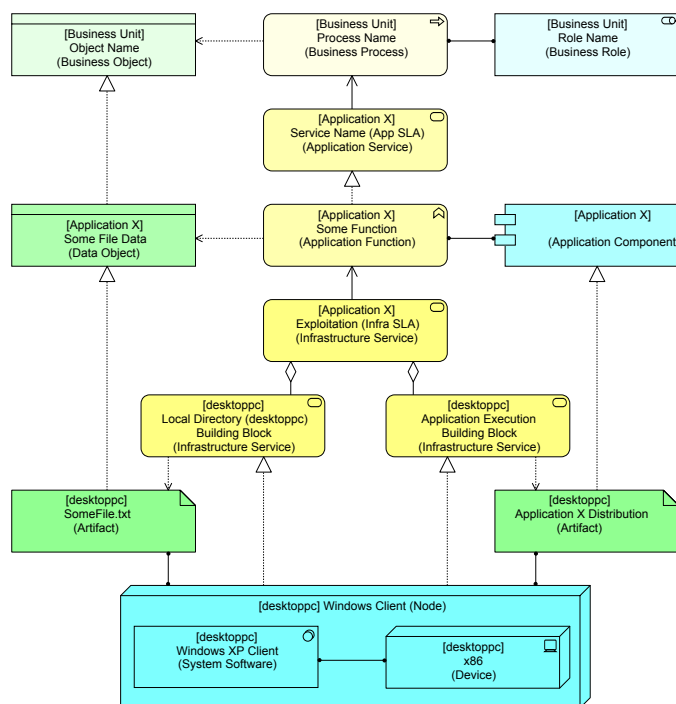
There is another slight disadvantage: experienced IT engineers might conclude from the use of the Aggregate relation that the 'Exploitation' Infrastructure Service does not need *all* its constituent parts to be up and running.

## 7.7 Application Deployment Patterns

We already saw a couple of basic deployment patterns (e.g. the PL\*SQL and BRE 'platforms') earlier in this section, Now, we are going to look at a series of examples that are pretty common.

### 7.8 Deployment Pattern: Standalone PC Application

We are going to start with the simplest: an application running on a standalone PC which can be seen in View 62:



View 62. Deployment Pattern: Standalone PC

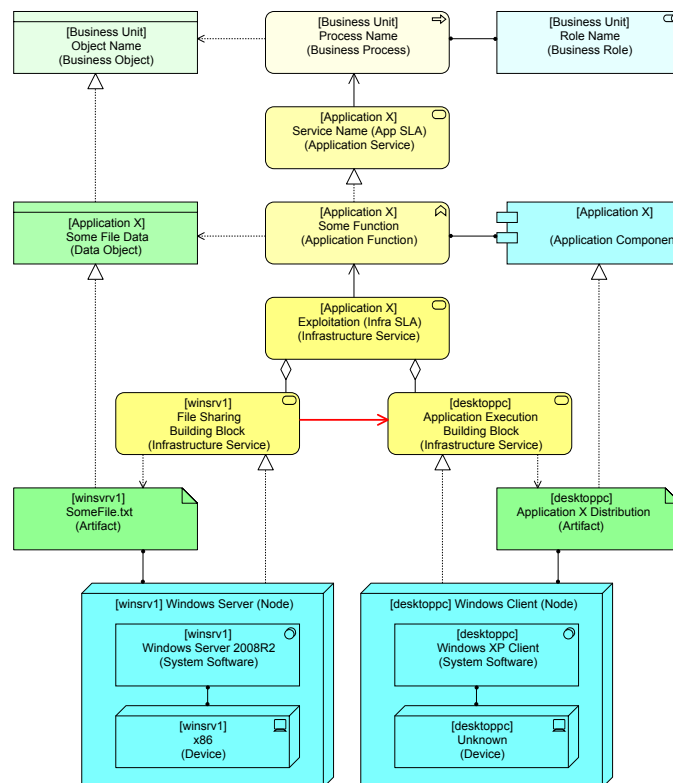
Here, a couple of common choices in my example deployment patterns can be seen again:

- I encapsulate TI in a Node like in View 57 on page 46. It is the Node and not one of its constituents (which are all composite parts) that is modeled to Realize the Infrastructure Service. In reality, it is the Windows System Software on the desktop PC that Realizes the Application Execution Service but we 'hide' those details.

- Any Infrastructure Service and its related Artifact are coupled with an Access relation.
- The Node is named after the Device that is embedded. In fact, this is often the Domain Name from the DNS system that gives access to the Infrastructure Service.
- The Infrastructure Services are named such that it is clear which Node realizes them
- As described in 7.6 "Infrastructure 'Building Blocks'", all Infrastructure Services needed to actually run an Application, including its data, are Aggregated into one abstract 'Exploitation' Infrastructure Service which has the name of the application in its label. The actual Infrastructure Services are 'building blocks' and the result is a service which a IT delivery organization could have an Service Level Agreement on.
- The Infrastructure Service Realized by the Node has been split in two: one for the actual running of the application and one for access to the data. In this example, that separation does not make much sense yet, but in more complex situation it becomes useful. So, the simple situation has been modeled slightly more complex than necessary here so it stays in line with the coming patterns.

### 7.9 Deployment Pattern: Standalone PC Application with File Server based Data

Next is a slight variation on the basic standalone pattern and in fact one which still should be pretty common in most organizations: a standalone desktop application, but with the data stored on a file server. Most organizations do not allow data to be stored on the local disk of a desktop, basically because that make backing up that data practically impossible. It can be seen in View 63.



View 63. Deployment Pattern: Standalone PC using a File Share

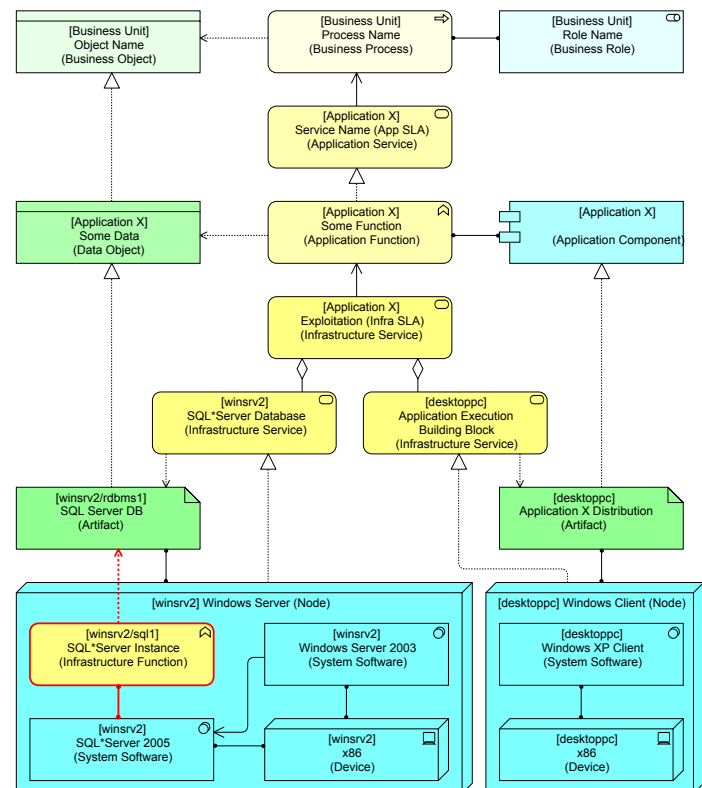
Not much changes here with respect to the previous pattern. Note:

- It is the Windows Server System Software that actually Realizes the File Share Infrastructure Service, but as in the previous example (and all coming examples) our pattern uses the Node as an encapsulation that hides internal structure and behavior.
- It is the Infrastructure Function Assigned-To the System Software on the desktop PC that actually uses the File Share Infrastructure Service, or in other words in this case: the Windows client mounts (in Windows lingo: maps) a remote file share under some drive letter. So, if I want to be exact, I need to draw a Used-By between the File Share and the Windows software on the PC. But as I want the Node to completely hide the internal structure, I have two choices:
  - \* Let the File Share Infrastructure Service be Used-By the desktop PC Node
  - \* Let the File Share Infrastructure Service be Used-By the Application Execution Infrastructure Service (which is a derived relation of the former one and the fact that the desktop PC Node Realizes the Application Execution Infrastructure Service. My choice is this one for two reasons:
    - If in a future pattern the Node Realizes multiple Infrastructure Services, not all of these may actually use what in this example is the File Share Infrastructure Service. Going via the Node may in the future create derived relations that are not there at all.
    - I keep the relations between the Infrastructure Services visible even if I do not show the Nodes in a view. Say, for instance, I model the “Internet” Infrastructure Service that my IT Department provides (as some of my systems may depend on it and I want to be able to take that into account), but I do not model the Nodes such as routers, switches and so forth that are needed to provide that services. The IT Department itself may model them of course. And in my own model, using a division of models in certain fixed ‘construction views’ makes this choice better, but we get into that in Section 15 “Construction & Use Views” on page 97.

So you see, my patterns do not always follow the exact and complete way the world is in core ArchiMate. For my role as Enterprise Architect, it is enough to have the relations as chosen now. You can of course choose differently.

## 7.10 Deployment Pattern: Classic Two-Tier Client-Server Application

The Classic Two-Tier Client-Server application consists of a client application that talks to a database (the server). It can be seen here:



View 64. Deployment Pattern: Classic Two-Tier Application

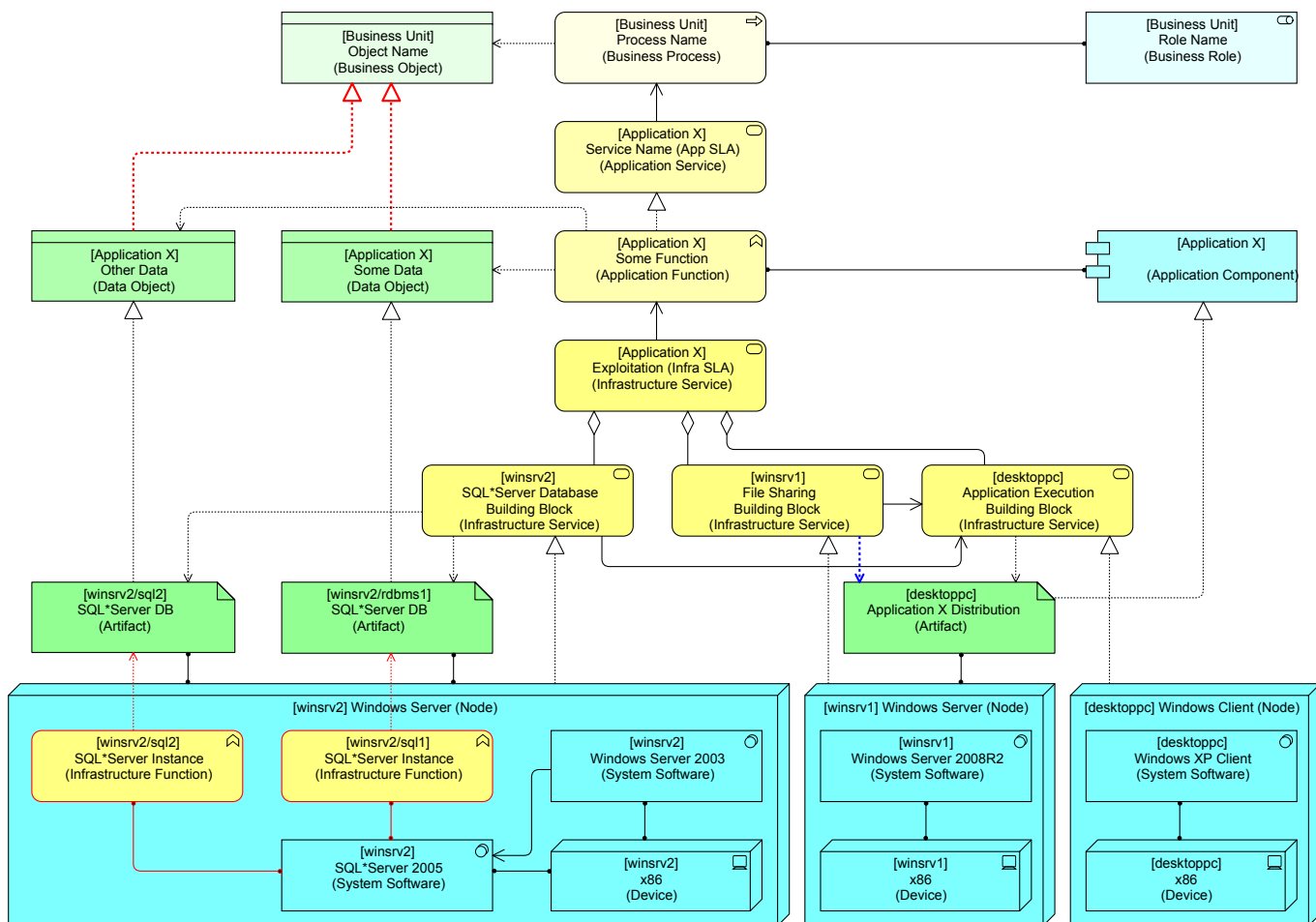
I have modeled a variation in red line/outline. This specific database system consists of independently running ‘instances’ (in operating system terms: it ‘forks’ multiple copies of itself). Suppose your large model is input for the CMDB and the infrastructure people need to know quickly what instance of the RDBMS system is at fault when there is a problem with an application. You can then choose to augment your database pattern with this information. In that case, you would have to model the instance inside the Node, while the Artifact is outside the Node. So, this augmentation breaks the basic idea of a Node encapsulating all information. I’ll repeat it again: you can create any pattern you want. The importance is that you stick to certain patterns if you want the model to be at its most usable.

The next pattern is a variation on the Classic Client-Server.

## 7.11 Deployment Pattern: Two-Tier Client-Server Application with mounted Client Application and two databases

If the client application in client-server is deployed on a different (file) server, the pattern becomes slightly more complicated, as can be seen in View 87 on page 53. In this example, I have created two ‘instances’ of the database software on the server and two databases. That was the variation that I talked about in the previous example.

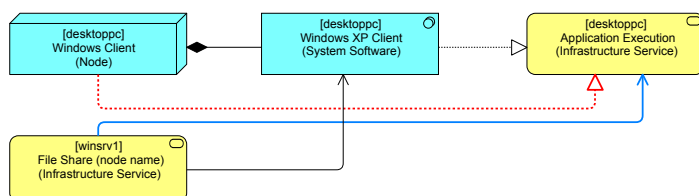
If you take a look at how the situation is in reality, you know that the *operating system* of the desktop PC mounts/ maps the file share on which the application is deployed.



**View 87. Deployment Pattern: Two-Tier Application with mounted client and two databases**

Formally, I would have to model a Used-By between the File Share Infrastructure Service and the Windows XP Client operating system's Infrastructure Function on the desktop. But I try to prevent as much as possible any relation to the inside structure of the Nodes. So, I create a derived relation in this pattern between the File Share Infrastructure Service and the Application Execution Infrastructure Service. The big advantage of this choice is that I do not have to show the Nodes (or worse: their internal structure) in a view to get the interdependencies.

Both derived relations I use to shield the inside structure of the Node can be seen in View 65.



**View 65. Deriving the relations from Infrastructure Service to Infrastructure Service (Used-By) and from Node to Infrastructure Service (Realization)**

That the desktoppc Node realizes its Application Execution Infrastructure Service (red) is the result from the fact that the Windows XP System Software is a Composite part of the desktoppc Node and the Windows XP System Software Realizes the Application Execution Infrastructure Service (via the omitted Infrastructure Function). That the File Share Infrastructure Service is used by the Application Execution Infrastructure Service (blue) is the result of the

fact that the File Share Infrastructure Service is Used-By the Windows XP System Software and the Windows XP System Software Realizes the Application Execution Infrastructure Service.

Note: the 'Application X Distribution' is Accessed by both the file share and the processing Infrastructure Services. This is correct, but you could leave out the blue one as you already have the dependency via the Used-By from the file share to processing Infrastructure Service.

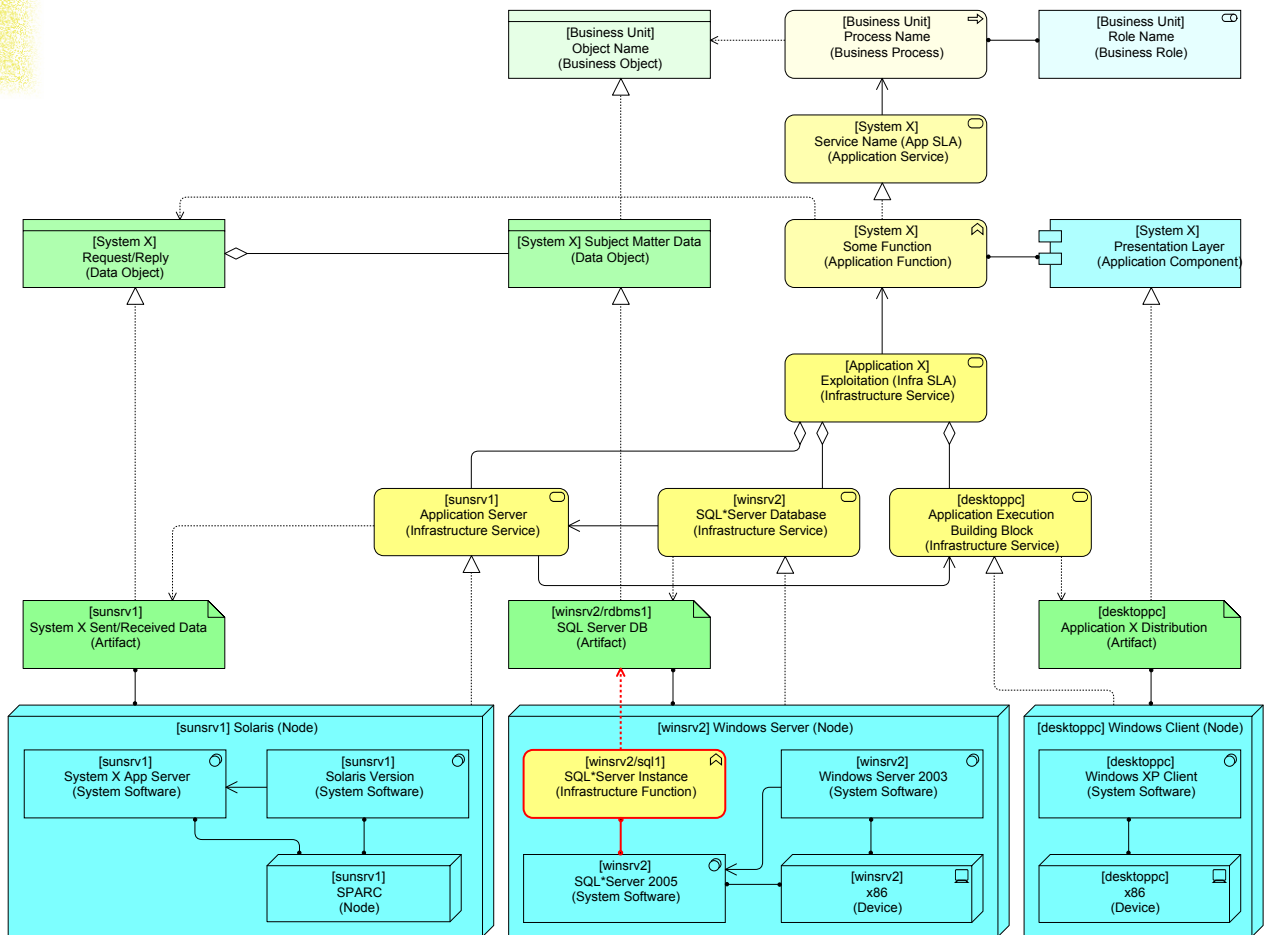
## 7.12 Deployment Pattern: Two-Tier Client-Server with a Remote Server (ASP)

The pattern in View 90 on page 55 is another variation on the Classic Two-Tier Client-Server:

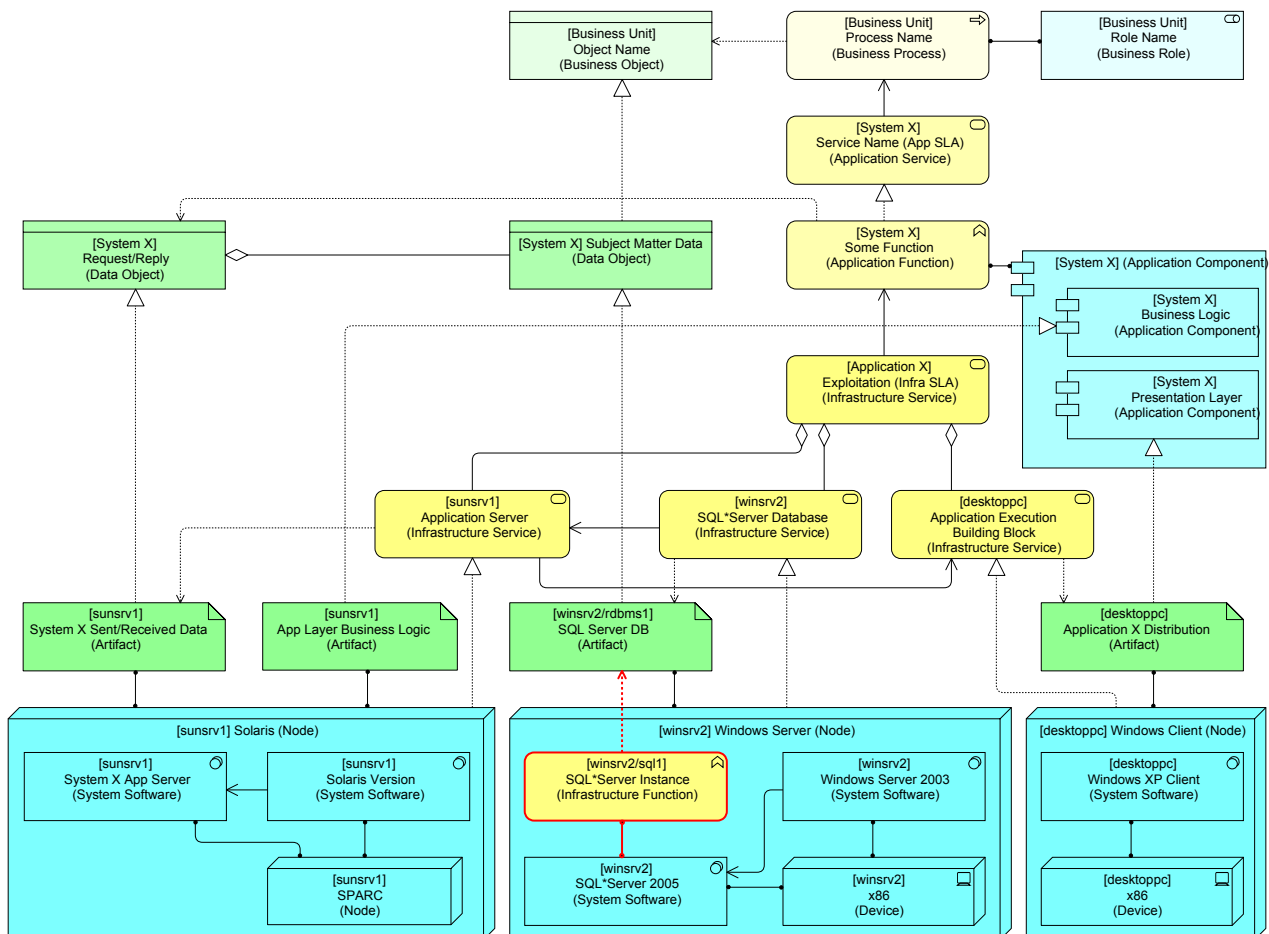
Here, the server is not your own, but it is provided by an external party. This is sometimes called an Application Service Provider, though the terminology for all these patterns is not very fixed.

This pattern might sound strange, but it actually happens. An example from Asset Management would be Bloomberg's AIM portfolio management and trading application. The Classic two-tier pattern was a client talking a database protocol to some database system. These days, the clients often talk to the server using the world wide web protocols (http or https) and exchange XML-formatted objects. But fundamentally, it is still the same. Another example would be Apple's iTunes application.





View 88. Deployment Pattern: Three-tier Application (with missing business logic)



View 89. Deployment Pattern: Three-tier Application

## 7.13 Deployment Pattern: Three-Tier Application

A Three-Tier application is an application where a thin client (often not more than a presentation layer without its own data) connects to an 'application server' where an instance of the 'second tier' application is launched specifically for that thin client. The second-tier application contains the actual business logic. This second-tier application itself then connects to the third-tier: the database layer. An first attempt can be seen in View 88. The fact that it has three tiers is nicely reflected in the interrelations of the Infrastructure Services (Used-By relations between Infrastructure Services). Modeled here is a very basic three-tier, where Tier 1 is an application in itself that is actually deployed on the desktop. These days, the first tier is often browser based, you can combine both patterns.

Now, in the normal Three-Tier case, the first-tier 'thin client' is generally nothing more than a presentation layer and having the actual Application Functions Assigned-To it is stretching it: it is just not true that the presentation layer performs the functionality. The actual business logic is after all in the second tier: the application server. When we looked at the language's 'two types of software' we introduced the notion that if it has business logic, it should be software in the application layer and when it is generic software it should be software in the infrastructure layer. We used that distinction already to set up patterns like the spreadsheet pattern and other 'platform' patterns.

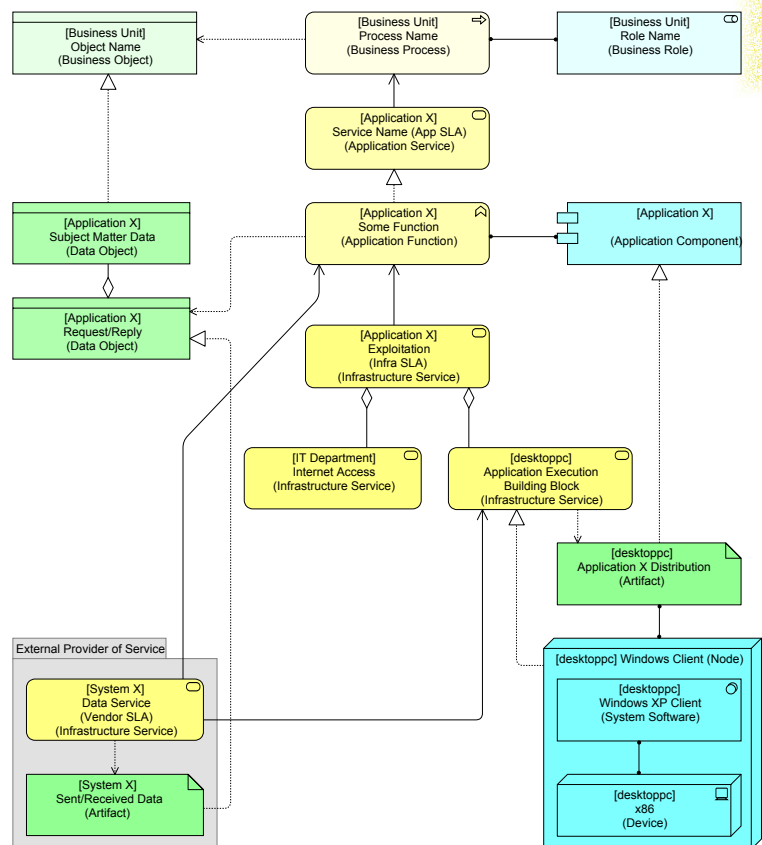
Personally, I like to see all the business logic in software at the application layer. In that case, I would propose another variation on the pattern, which can be seen in View 89. That business logic that is deployed on the application server is now visible as an Application Component (the business logic). The thin client is also an Application Component (the presentation layer). Together they make up the system. It is quite a large net we are casting here, because we end up creating a composition of Application Components that in the deep down technical IT reality can hardly be seen as one 'program' at all. But on the other hand: the entire system's functionality that provides the service for the business is indeed a combination of presentation layer and application layer. In Section 8.2 "Application Collaboration is an Anthropomorphism" on page 63 we'll see a variation using Application Collaboration.

**Question 1.** What Style Guide are we breaking in View 88 and View 89?

**Answer 1.** Switching position of Application Tier and Data-base Tier leads to less line crossings

## 7.14 Deployment Pattern: Software as a Service (SaaS)

Another variation is a pattern that is often called 'SaaS' for 'Software as a Service'. It is both a variation on the two-tier pattern and the three-tier pattern. The difference between this pattern and 'Two-Tier Client with a Remote Server' (ASP) is normally that an internet browser is used as a plat-

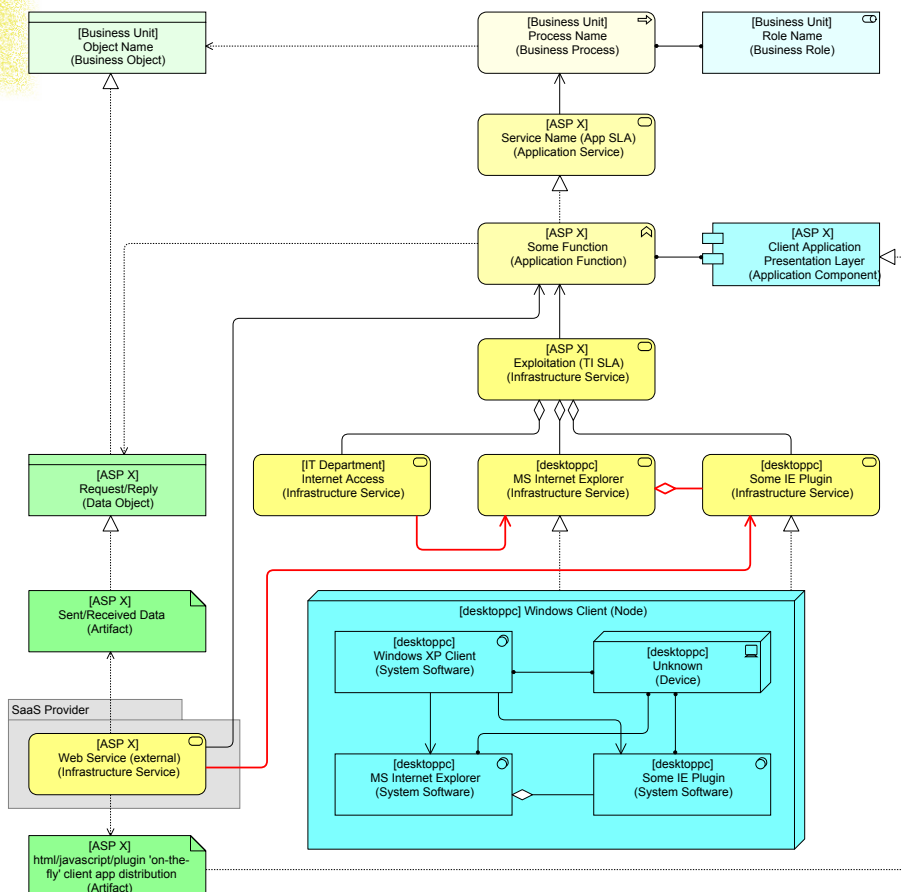


View 90. Deployment Pattern: Two-Tier Application with remote server

form for a very thin client to access the software. As we saw earlier, when we were discussing the browser-based application, the Application Component here is created on the fly from the html/javascript/etc. Artifact sent by the web server. So, in SaaS, the outside provider provides not only the server, he also provides the (Javascript) 'presentation' application on-the-fly and we just provide the platform that that application needs to run. In the example in View 91 on page 56, I made matters a bit more interesting by modeling a SaaS provider which demands from us the installation of a certain plug-in for the browser (e.g. Silverlight, Flash or some other proprietary plug-in). I have used an Aggregation relation (next to the Used-By) for the interdependencies of Infrastructure Services, to point out that the relation between the browser and its plug-in it *loads* is much stronger than the relation between the browser and the other services (Internet access, remote web server) it *uses*.

This pattern shares with the initial three-tier pattern, however, that the application that runs locally in the browser platform is nothing more than a presentation layer. The business logic runs not locally, but remotely. This has the rather unpleasant consequence that we have to Assign the Application Component representing the *presentation* layer to the Application Function representing the *business* logic.

Therefore, we can better turn our original perspective (our IT provides services to our business) on its head. It is not *our* IT that provides the Application Service to our business, it is *their* IT, it is SaaS after all. We just offer *them* Infrastructure Services (a platform for the thin client and an



View 91. Deployment Pattern: Software as a Service (SaaS) - variation 1

internet connection) so that *they* can offer *us* the Application Service as can be seen in View 92 on page 56.

This pattern makes it very clear that the external party is providing an Application Service to our business. Our Business Objects that are Accessed in our Business Process do not even have a representation at the application layer as they only exist *outside* our architecture in the 'IT cloud'. Of course, when that same SaaS-provider offers a way for us to get a copy of the data, we can model that too in addition to what is shown above.

## 7.15 We only model what we can change

You might wonder why I did not include the Application Functions and Application Components in the SaaS-variant in View 92. The reason is a sort of golden rule: "we only model what we can *change*". Modeling the internal structure of how your SaaS provider functions internally (something sometimes found in its documentation) is not only unnecessary, it distracts you from how *you* as an organization are functioning, which is what your Enterprise Architecture is all about. If your organization has outsourced its IT in a SaaS-manner, modeling data objects and infrastructure of the SaaS-provider gives a false picture of your Enterprise Architecture. It is outside your scope.

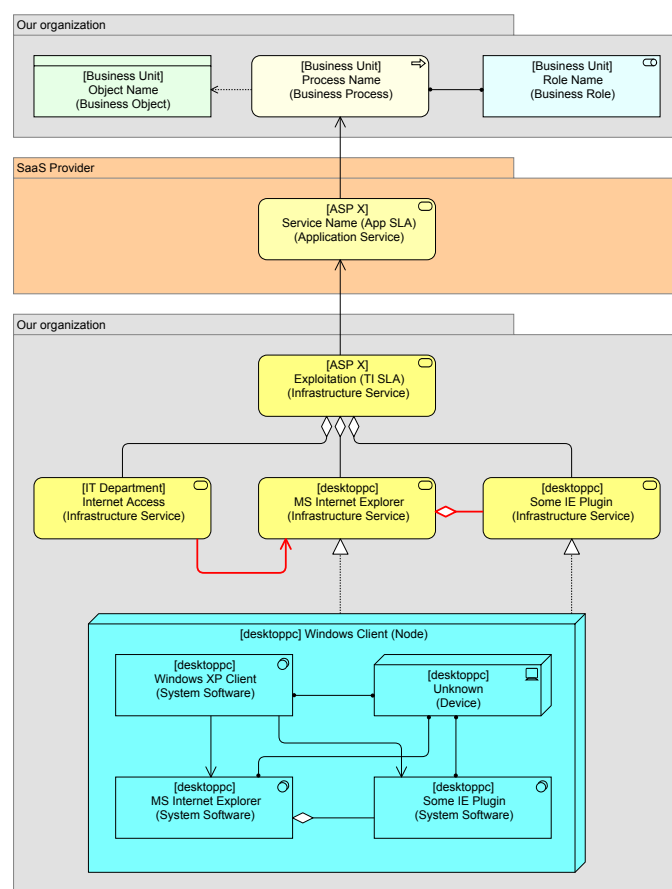
So, I suggest to draw the line for modeling at what you actually directly can see from the perspective of your own organization. Everything you model, should be part of what is under the control of your organization. And you should be able to check your model against your own reality to see if it is correct.

Now, there are reasons why you sometimes need to know more. For instance, regulators may force you to be aware of the quality or other aspects of your provider. You might have to be able to show to your regulators that you are in 'control' on aspects as security and such. Insofar as you want to support that with ArchiMate models, I suggest you keep that separate from your own current state model.

## 7.16 Deployment Pattern: Providing a local ASP

An ASP/SaaS pattern can be used inside your company as well, in case you use application servers internally. For instance, when you deploy Microsoft Dynamics, you end up setting up a Microsoft IIS Web Server, within which environment you deploy second-tier applications you build, which then again talk to an underlying Microsoft SQL\*Server RDBMS. In View 93 on page 57, you can actually see what is going on under the hood of the ASP/SaaS pattern, because you are providing it to yourself. If you look closely, you'll see that the pattern looks a lot like the

three-tier pattern in View 89 on page 54, but without the third tier. This is no coincidence, because the ASP pattern is just like the three-tier pattern, with the only differ-



View 92. Deployment Pattern: SaaS Provider uses our Infrastructure



ence that the presentation layer (the first tier) runs in the browser *platform*.

Of course, the reality of your Local ASP is far more complex. Say, a reporting server is used to create reports every day. And your Local ASP based solution has to send out mail as well to alert people in certain circumstances. And of course, there is a Third Tier: that data must be stored in a database. Or, if the second tier calls upon the reporting server for some functionality we actually have sometimes *four* tiers: thin client → application server → reporting server → database server. The extended example can be seen in View 94 on page 58.

**Question 2.** Which relation is missing in View 94?

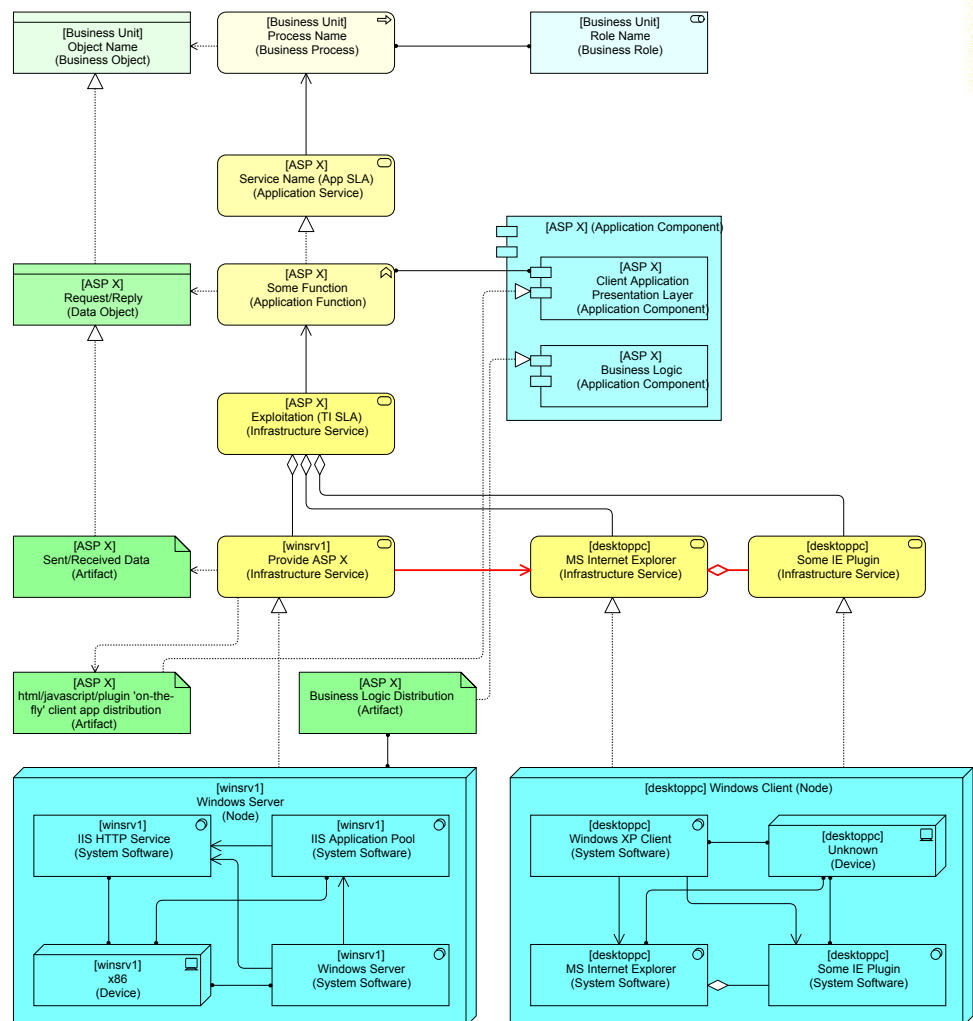
**Answer 2.** The Assignment from the Business Logic Artifact to Node winsrv1

## 7.17 The Use of Patterns (reprise)

We will now move to a couple of TI-specific patterns that have to do with types of infrastructure clustering. Infrastructure clustering is generally used to improve performance or reliability.

Now, so far we have seen pretty simple TI-structures, starting with View 57 on page 46. We had Nodes that embedded a single Device and the System Software that was Assigned-To that Device. Then, we Realized the Infrastructure Service directly from the Node, making the Node (and the Infrastructure Service and the Artifact) a sort of cut-off between Infrastructure use and Infrastructure structure. The Node is used in fact as a sort of object in an object-oriented approach, hiding its internals.

This approach in combination with the Building Blocks approach (View 86 on page 50) creates fixed routes between TI and applications. For instance, if you want to know which Application Function depends on a certain Device, you follow the following route: Device is Composite part of Node, which Realizes Infrastructure Service which is part of an Aggregate 'Exploitation' Infrastructure Service which is Used-by an Application Function. This is not a valid route for a formal ArchiMate 'derived relation', but when you use the patterns consistently, this route will always give you the Application Functions that depend on the device. Such consistent pattern use in your modeling will give you (depending on your modeling tool) all kinds of useful (and more importantly: reliable) automated analyses. The less discipline while modeling, the more variation in your patterns, the more difficult it becomes to define 'standard' dependency routes and use your model



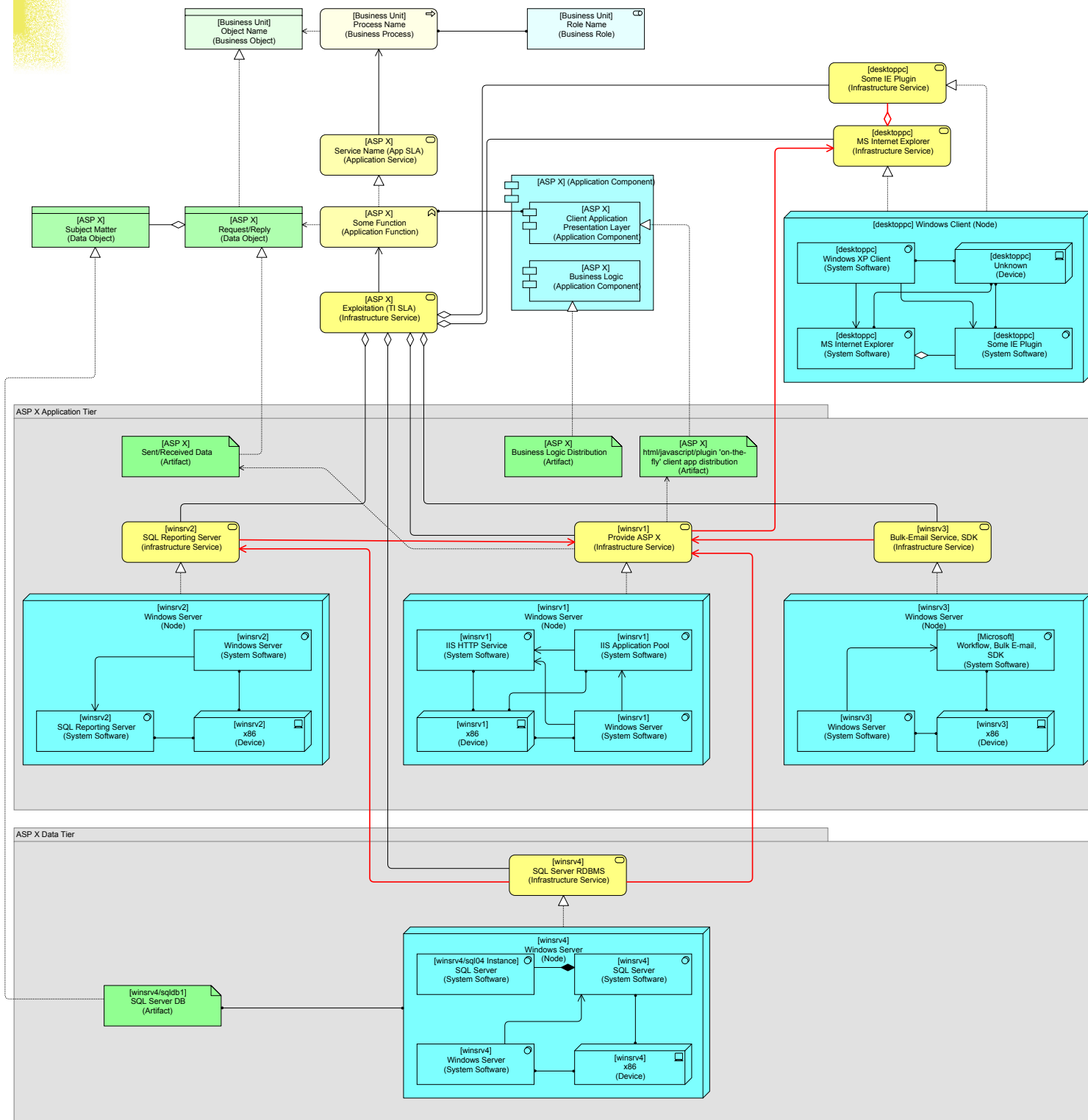
**View 93.** Providing your own ASP

effectively. ArchiMate is not just a language or a grammar, it is a *modeling* grammar, and if you combine a wild selection of modeling styles and patterns, the result becomes less usable. If you do not use patterns at all, you're at the mercy of painstakingly letting a human (probably yourself) analyze the model, something that may overwhelm you if the model becomes large (which it does if you want to model your Current State).

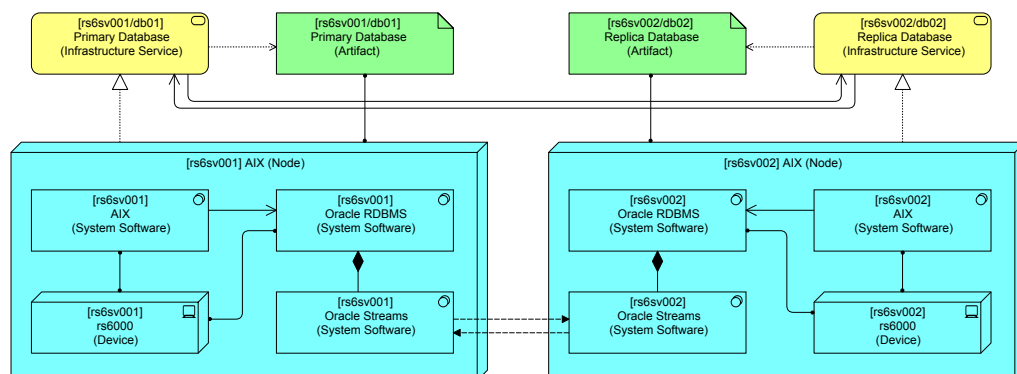
However, we will now get into a bit of IT complexity where we might (for sake of aesthetics for instance) decide to introduce alternative patterns. E.g. another level of Node embedding or another level of Infrastructure Service Aggregation. When we do this, it is important to remember the price one pays: for every alternative pattern, you need a second route if you want you standardized (and thus including automated) analyses to be complete.

## 7.18 Infrastructure Pattern: Database Replication

The first example of a very lightweight sort of 'cluster' is database replication and it can be seen in View 95 on page 58. Actually, this is not really a cluster if one defines a cluster as a combination of items that present themselves as a *single* item to the outside world. In database replication, the main database replicates all its actions to a second read-only database. This is above all useful for performance. Suppose you want to run complex management summary reports on your core system's database, such



View 94. Providing a local ASP with all the details



View 95. Infrastructure Pattern: Database Replication





## 7.19 Infrastructure Pattern: High-Available Database Cluster

If we create a true cluster from multiple databases, they will behave as a ‘single entity’ to the outside world. That is, they present a single Infrastructure Service to the outside world and the outside world does not know which actual database is being used. An example is modeled in View 96. In this example I have modeled a database cluster that provides two databases. I have also put in a little detail about the technical situation. Server rs6sv001 has two databases: rdbms11 and rdbms21 (Assignments in red). Rdbms11 is the rdbms1 database on the first server. Rdbms21 is the rdbms2 database on the first server. In fact only the rdbmsxy databases exist physically, The rest are just aliases in the database system. Rdbms1, for instance, is the alias for either rdbms11 on rs6sv001 or rdbms12 on rs6sv002. The setup in the example uses the second server for the first database as first server for the second database and the first server for the first database as second server for the second database (you might need to read this sentence multiple times, in short: they are each other’s fall back but also provide their own primary database). So, it is a pretty efficient setup. The fall back server is used and not just idling expensively in the background. Of course, this requires that — in the case of a failure of either of the Nodes — the applications can do with roughly half the database performance. As extras I have modeled the upper Artifacts. They are aliases in the database’s naming scheme. They may be used by some applications and we therefore need them in our model. The nodes also contain a batch-job agent. Both devices need such an agent, e.g. to start jobs on them during the night (e.g. a backup job for rdbms2 on rs6sv002 and a backup job for rdbms1 on rs6sv001).

Now, applications that access this cluster always go to ‘rs6sv001’. For the outside world, that is the name of this cluster. In this specific example, the Oracle RDBMS has its own database-name-to-system mechanism that handles this (as far as I know, I’m not an expert) This is modeled as an extra Aggregation layer of Infrastructure Services.

Variations are of course possible. We could for instance have chosen to model only one abstract Node. But that would have meant we lose the information on which Device the actual physical databases are, unless we would have Assigned them to the Device instead of the Node. Or we could have modeled an extra layer of Node-embedding and just have the top Infrastructure Services being Realized by that outer Node.

## 7.20 Infrastructure Pattern: High-Available Server

The third example of clustering is a high-available operating system or server. Here, two devices run an operating system and on top of that they run software that makes them act as a single system. So, we

can approach the cluster as a single system where we can for instance install software on. In the example in View 97, we see a Node that contains two RS6000 Devices running IBM’s AIX. On top of AIX, HACMP software (HACMP stands for “High Available Cluster MultiProcessing”) has been installed which delivers a HACMP Service. These two services together Aggregate into a service that provides a single High-Available system as an internal service of the abstract Node. In this example, Tibco has been installed on the HACMP Cluster. Tibco itself is unaware that there are two Devices and two operating systems it runs on: it sees only a single operating system.

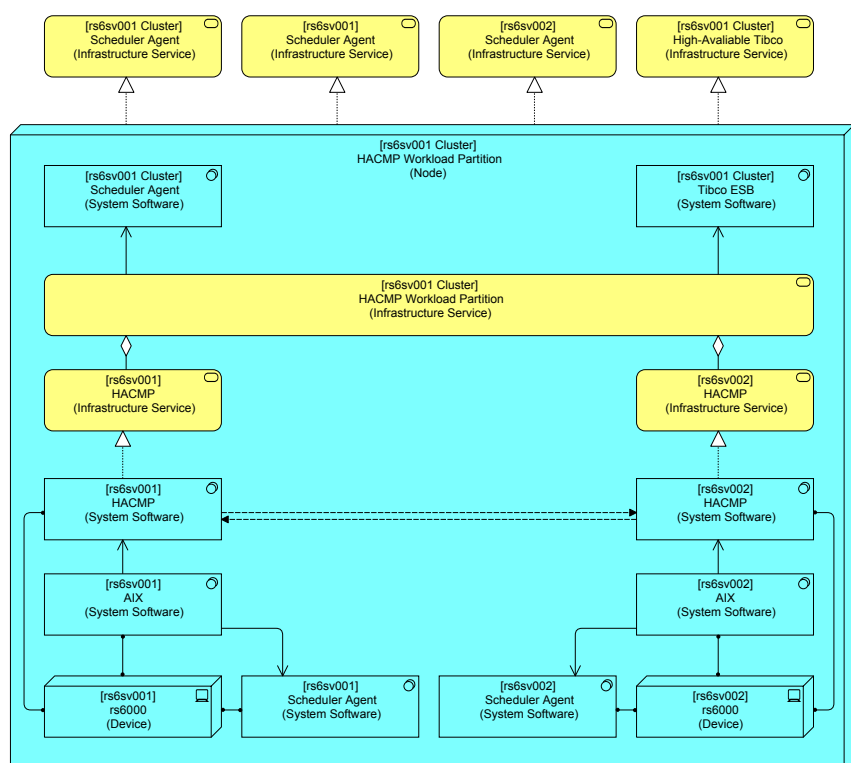
I have also modeled three installed scheduler agents here. One on the operating system of each device and one on the cluster. If, for instance, a batch job for Tibco needs to be started (e.g. a scheduled restart), the scheduler must use the “[rs6sv001 Cluster] Scheduler Agent (Infrastructure Service)”, but if for instance some housekeeping job on one of the underlying devices is necessary, it should use “[rs6sv001] Scheduler Agent (Infrastructure Service)”. Not visible in the view: the Tibco and ‘cluster’ Scheduler Agent System Softwares have been Assigned to the Node.

**Question 3.** Can you spot the incorrect ArchiMate in View 97?

**Answer 3.** You cannot officially nest the ‘HACMP Workload Partition’ Infrastructure Services inside Node.

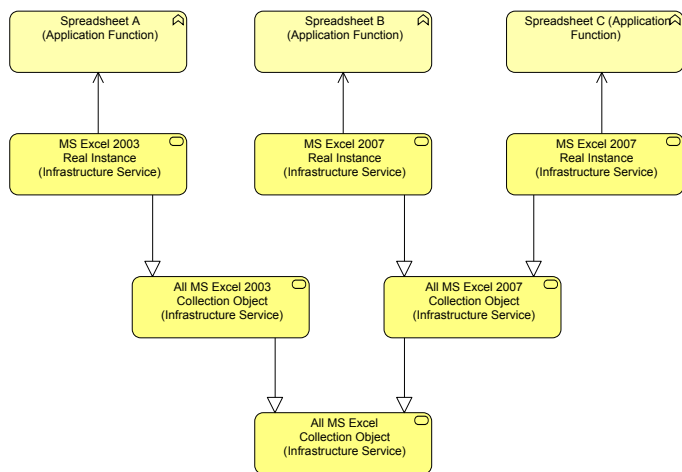
## 7.21 Using Collections and Abstraction in a model

When you make large models to describe your current state architecture, it tends to become useful to have certain groupings in your model. I am not talking about grouping in a view, but grouping objects ‘together’, e.g. via Com-



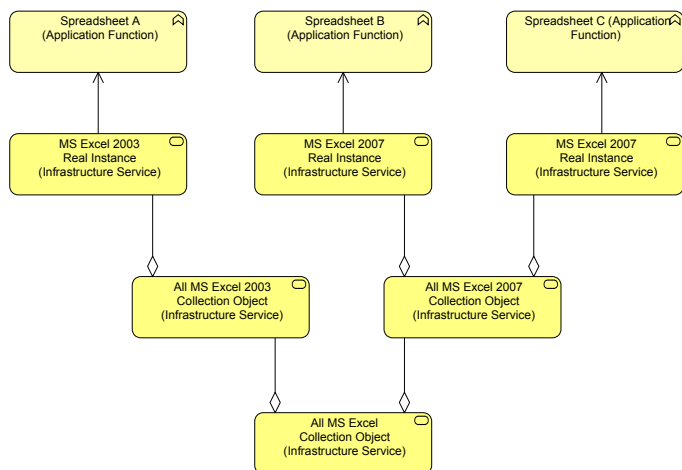
**View 97. Infrastructure Pattern: High Available Server.** Set up pioneered by colleague/hired gun Roy Zautsen.

posites, Aggregations or Specializations. For instance, if you have multiple versions of MS Excel in your landscape, you might want to group all instances of Excel 2003 and Excel 2007 in your model so it is easy to find them when for instance you have to do an analysis that has to do with upgrading one or the other (e.g. which business processes are affected when I will upgrade all MS Excel instances to MS Excel 2010?). You might think of doing it like in View 66:



**View 66.** Adding collections to your model by Specializations

Here abstract objects have been created to stand for the different types of Excel we have in our landscape. After all, the 'real' MS Excel 2007 on a PC somewhere, is an instance of a generic type 'MS Excel 2007', which in turn is a subtype of 'MS Excel'. (Incidentally, most organizations would love to be able to know all spreadsheets that are used in their business, but few do). A different approach is using an Aggregation as in View 67:



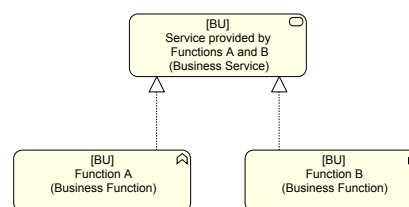
**View 67.** Adding collections to your model using Aggregations

This looks almost the same. The first has the advantage that it is using a true abstraction relation for the abstractions: in other words: these generalizations do not exist in reality, they are *types*. Hence, using a specialization is 'proper'. The second looks not so much at a Type/Class, but sees all those Excels out there as a large *collection* of real instances. This is not a real 'abstraction', but it is 'proper' too. The second has the advantage, though, that derived relations from the abstract objects to the real objects are possible: 'Spreadsheet C' uses 'Excel'.

You can probably get pretty heated discussions amongst architects over one or the other. But if your tool does not restrict your analyses to only derived relations, they both work out equally in practice. In practice I have settled on using the second because of the following unlikely scenario: If my tool ever in a new release restricts me to doing analyses through proper ArchiMate derived relations, that pattern will still work.

## 7.22 Concurrent Realization?

In ArchiMate, a function (or in the business layer, a process) can Realize a service. Nothing in ArchiMate stops you from having multiple functions all Realizing the same service. But what does that mean? Generally, it is meant by those that use the pattern that both functions together Realize the service. This construct, however, is somewhat problematic. Let's start with the example in View 68:

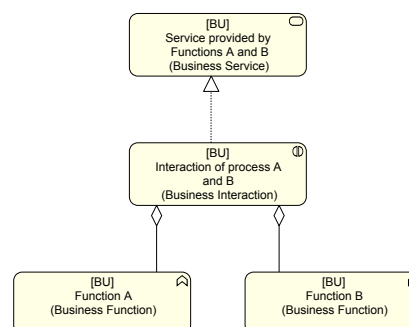


**View 68.** Two Business Functions Concurrently Realize One Business Service

This is what you often see in a model and it is intended to mean that both functions together realize the service and not 'either of them can realize the service', though I can imagine situations where for instance three locations of a function all are modeled separately (as they are performed by different roles that are fulfilled by different actors) and all three perform *exactly* the same service to the outside world. But back to the commonly intended meaning, of two different processes that *together* realize a service. Now, if these two functions provide the service together, we have two options:

- Both functions work together to provide a single unified service.
- Both provide a sub-service of the overall service.

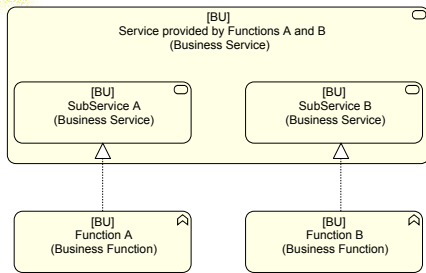
The first, however, should be modeled using Business Interaction (and if active components are added: Business Collaboration):



**View 69.** Two Functions Interact to Realize a Service

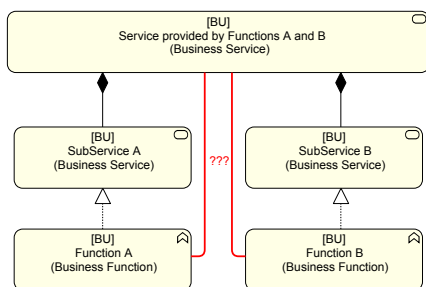
Note that we do not strictly follow ArchiMate here as ArchiMate says an Interaction is the behavior of a Collaboration and not so much an Aggregation of functions or processes like a collaboration is an Aggregation of Roles

or Application Components. But this is in the spirit of ArchiMate. And the second looks like View 70:



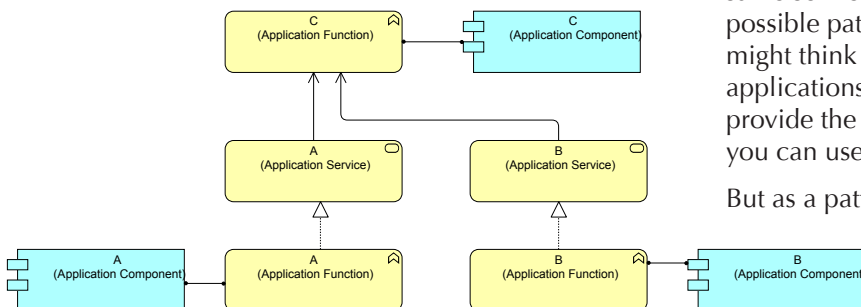
**View 70.** Two Business Functions each Realize a Sub-Service of an Overall Business Service

If you expand the Nesting of View 70, it looks like View 71 (ignore the red associations for now).



**View 71.** Two Business Functions each Realize a Sub-Service of an Overall Business Service, expanded

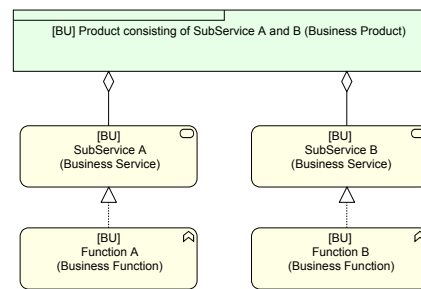
Now, the question is: what is the relation between Function A (or B) and the 'Service provided by Functions A and B'? In the image, these relations are depicted by the red lines.



**View 99.** Application C uses Application A and Application B

Well, since the Realization between the Function and the corresponding Sub-Service has a direction opposite to the relation between the Sub-Service and the overall Service, there is no official derived ArchiMate relation. And if you look at dependency in a certain way, it is also logical: Sub-Service A does not depend on Function B. So, if we have two independent sub-services, we should not model a Realization between each function and the overall service because *they do not exist*.

In the case of a collection of independent sub-services that are offered to the outside in a 'package', you can probably better use the Business Product object as in View 72:



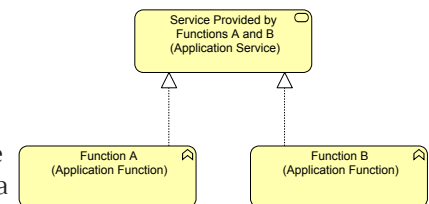
**View 72.** Two independent (Sub-)Services make up a single Product

So, summarizing:

- If an overall Business Service that is truly realized by two or more functions (or processes) in collaboration/interaction, you can better model the true collaboration/interaction.
- If it is not so much function A *and* function B Realize a Service together, but function A *or* function B Realize the *same* Service *independently*, we should use the pattern in View 68 on page 61.
- And if functions A and B Realize Sub-Services for an overall Service or Product, we can best use View 70 or View 72

Realizing a service by two functions is also possible at the Application Layer, of course, as can be seen in View 73.

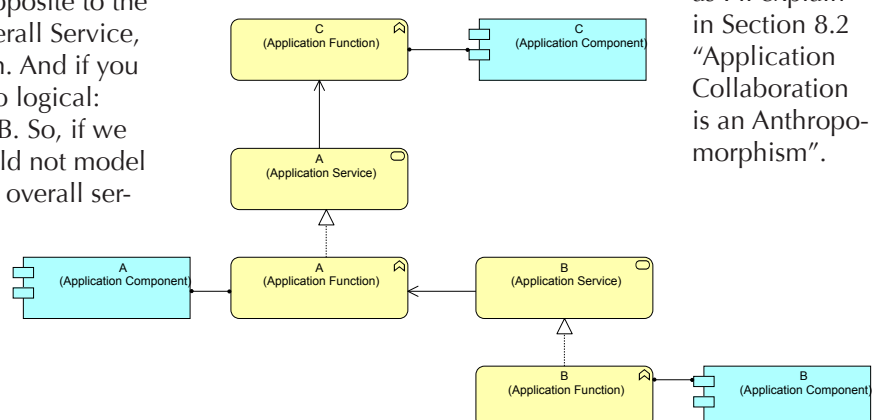
Here too, you need to wonder what is being meant. If two independent functions can provide the same service, this is a possible pattern. You might think of two applications that both provide the same and you can use either of the two to do what you need.



**View 73.** Application Service Realized by Two Application Functions

But as a pattern showing two functions Realizing a service together (which is how you will see most modelers using it), you run into the same sort of questions as above. Of course, ArchiMate also offers the Application Collaboration and Application Interaction as an object to model the way two Application Functions may offer a single service. I personally think this should be avoided in your models,

as I'll explain in Section 8.2 "Application Collaboration is an Anthropomorphism".



**View 98.** Application C uses Application A which uses Application B



## 8. Anti-patterns

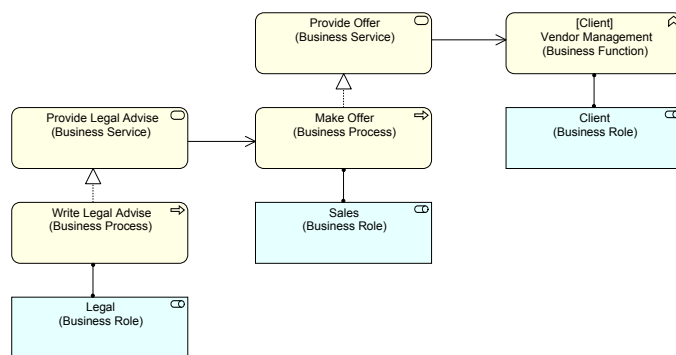
### 8.1 Multiple 'Realizers' for a single Service

As we saw in the previous section, we can say that in most cases, using multiple Realizers for a service is not a good plan.

### 8.2 Application Collaboration is an Anthropomorphism

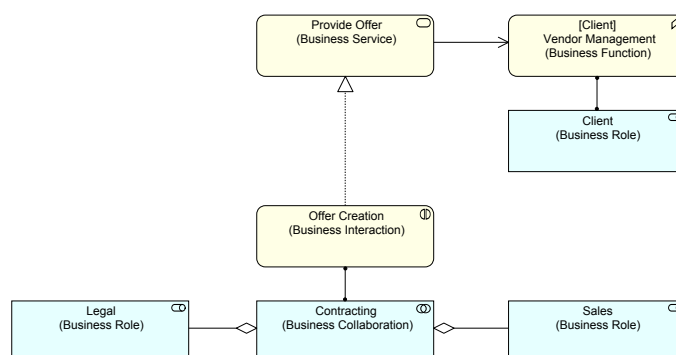
The designers of ArchiMate added two collaboration concepts to the language: a Business Collaboration and an Application Collaboration. In Section 10.2 "Business Function or Business Process?" on page 72, we will see that Business Collaboration has an equivalent in the real (business) world. But this is hardly true for Application Collaboration (though with an exception, see below)

In ArchiMate there are two basic 'cooperation' patterns. One is Used-By. A behavioral element (process, function) can use another behavioral element (service). We repeat View 20 on page 23 here in View 74.



View 74. Collaboration: Sales Uses Legal on Order Creation

The other is a Collaboration. We repeat View 21 on page 23 in View 75.



View 75. Collaboration: Sales and Legal Collaborate on Order Creation

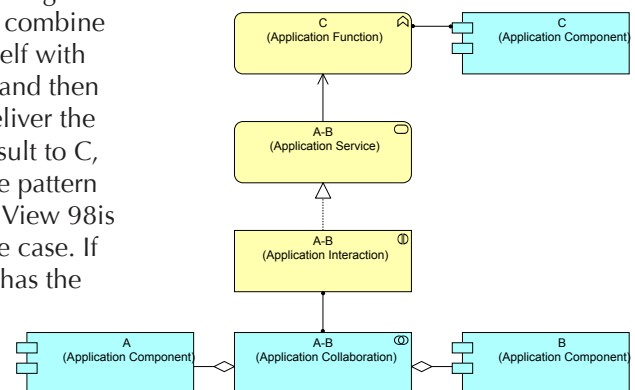
In the business layer this is both an understandable way to look at it. But when do you choose one, and when the other? If you look at the concept of 'collaboration' from a human perspective — which is OK at the business level — it is about decision making. I would go for Collaboration when both processes must actively negotiate to make decisions *together*. An example is the 'exception collaboration' mentioned in Section 10.2 "Business Function or

Business Process?" on page 72. I would use Used-By when the process or function used has no real say in the matter, the decision power lies all with the 'user' and not with the 'used'.

If you try to imagine how a real Application Collaboration would work, you have to differentiate between the two uses of Used-By (see section 1.2 on page 17). One is an application used by a business process or business function, the other is an application used by another application. In the latter case, the question is: which application provides the service that the 'user' application connects to? Basically, there are two options:

- The 'user' application C uses another application A which itself uses yet another application B as can be seen in View 98 on page 62.
- The 'user' application C uses both application A and application B as can be seen in View 99 on page 62.

The difference between these two is where the application logic is that *combines* the functionality of A and B. If A has logic to combine itself with B and then deliver the result to C, the pattern of View 98 is the case. If C has the



View 100. Application C uses a Collaboration of Applications A and B

logic to combine the results of A and B, then the pattern of View 99 is the case.

Now suppose we would draw this using an Application Collaboration as can be seen in View 100. It neither fits the first nor the second option. It introduces objects that are abstract and that do not exist in reality. It certainly does not fit the second option: the actual interaction does not take place outside of C, it takes place *inside* C. And in the case of the first option, it suggests an equality between A and B that is not there.

So, in the case of applications using a service provided by two other applications, we shrink the amount of objects from 8 to 7, but at the price of introducing 3 objects that do not exist in reality and that muddy the picture of what is really going on in a substantial way.

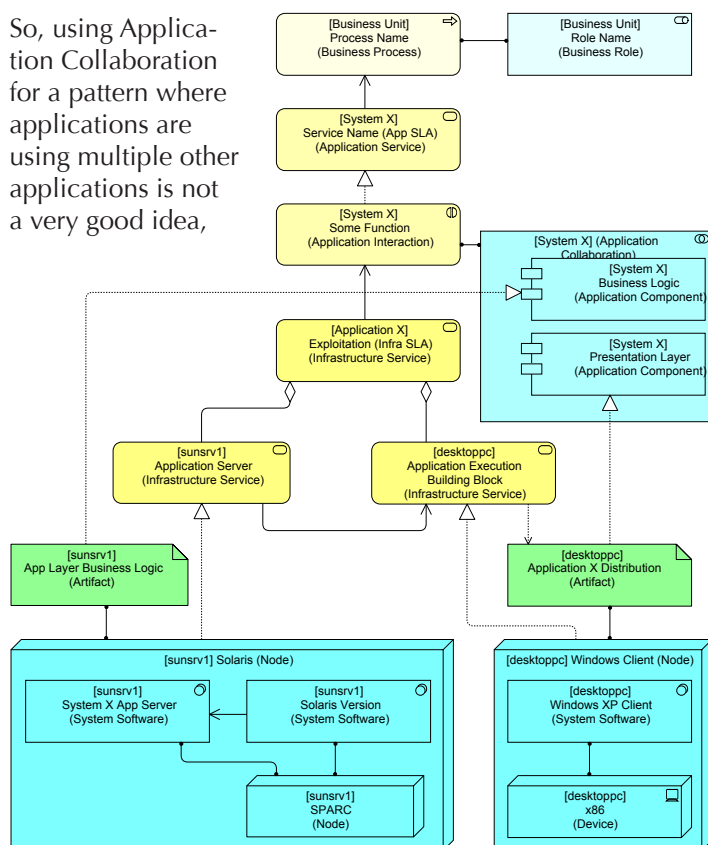
And that is not all: try to add the Application Interface to the mix and you get View 101 on page 64. Given that an application can only use *one* interface for a service, we now also have introduced an interface that *cannot* exist in reality. In terms of creating a good model of what is going on, matters have gone from bad to worse. To make matters

worse, if you use an Interaction to describe behavior (and properly document the Application Interaction object), you will still need to fully document the separate behaviors that make up the Interaction if they are used elsewhere in solitary form or in another Application Interaction. The same documentation will be in different places and it will be difficult to keep it in sync.

Is it possible to think of a pattern where Application Collaboration makes sense when the 'user' is another application? Yes, but it is pretty convoluted. Here is an example:

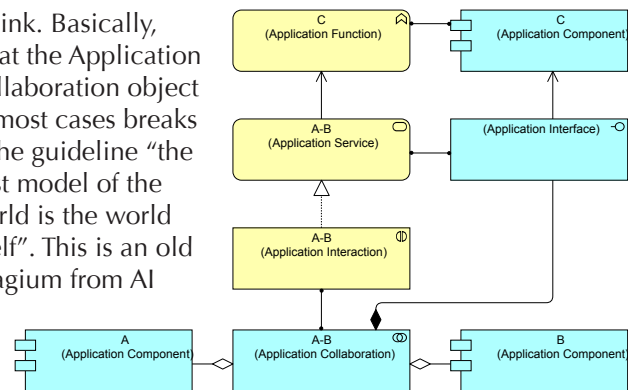
Suppose there are multiple applications that provide the same service, say a search service. A request for such a service is given to all that can provide the service. All possible search providers calculate a quality of the answer they can provide: how much 'trust' they have in the answer they can give. All write this 'trust' value in a common data object. After a while, all look in the common data object if they had the highest trust value and only the one who had the highest trust value delivers the service. Basically, what is described here is a sort of AI with autonomous software agents: on the basis of internal rules (which they all share but have implemented separately, which in this case is possible in a practical sense because the rule is very simple) the applications amongst each other decide who is going to provide the service. Given that all applications participate in making that decision, you can argue that there is a single application function that is in effect distributed across multiple application components: hence a collaboration of all applications is assigned to that single function. The interface for that solution will be a nice piece of engineering too. In practice, software engineers will most likely design a 'master application' and the architectural relation reverts to Used-By of View 99 on page 62.

So, using Application Collaboration for a pattern where applications are using multiple other applications is not a very good idea,



**View 102.** Using Application Collaboration to model a multi-tiered software system

I think. Basically, what the Application Collaboration object in most cases breaks is the guideline "the best model of the world is the world itself". This is an old adagium from AI



**View 101.** Application C uses a Collaboration of Applications A and B, with Application Interface

where they learned over a couple of decennia that creating abstractions generally decreased usability. The same is true for Enterprise Architecture modeling, especially when you are busy modeling your Current State or the end state of a Project. I do not think using such abstractions is always wrong, they are perfectly reasonable. Enterprise Architects, being creatures that love the simplification that comes with abstraction, especially love it. My problem with too much abstraction in your Enterprise Architecture models is that it is pretty unmanageable, just like in AI. Stay close to what is really there and less confusion and more usability is your reward.

Anyway, what about using Application Collaboration for an Application Service that is used by the business and that is thus described more in business (human) terms? Well, an Application Service for the Business is of course defined in business terms and as such, a collection of services thrown together is possible from a human point of view. But think again of the interface which messed things up before. Each of these applications has its own GUI. Using an Application Interaction again forces you to create a non-existing (abstract) Application Interface that under water consists of the two separate GUIs collected in one.

All in all, you're generally better off by not using multiple realizers for one service at both the business layer and the application layer. It looks simple, but it also introduces many untrue aspects in your model, so in many cases that will be a high price to pay.

But there is an exception.

Recall the multiple-tier application deployment patterns like for instance View 89 on page 54. There the presentation layer of the multi-tier application and the business logic layer each are an Application Component Realized by an Artifact. In that View, the complete system was modeled as an Application Component with each of the Artifacts Realizing a sub-component. And while technically it is the presentation layer that *Uses* the business logic layer, it is quite nice to model this as a true Application Collaboration as my colleague Jos Knoops proposed. This is shown in View 102.

I must admit I rather like this use of Application Collaboration. We also decided not to use it, though it is more aesthetically pleasing. The reasons is that it meant we had to change our analysis viewpoints such that everywhere the pattern is to look for an Application Function, we would also have to model the alternative. But we might still do it, because it is also nice to have a way to differentiate be-

tween single- and two-tier applications where the business logic resides with the presentation in one component, and three-and-more tier applications where the business logic is spread out over multiple components working together.

**Question 4.** If you want to add an Application Interface to '[System X] Service Name (App SLA) (Application Service)', of which Application Component is it a Composite child?

**Answer 4.** '[System X] Presentation Layer (Application Component)' and derived from the '[System X] Application Component'.

### 8.3 Using the Association Relation

The Association relation has a couple of official uses in ArchiMate. As the Association relation is allowed between all concepts, it is a kind of catch-all. It is always allowed and it is always possible.

As far as I'm concerned, most of the time, the Association relation is for wimps. It is a sign that you haven't thought out what the real relation is. It is often a sign of sloppy modeling. So, here are my rules regarding the use of the Association relation:

- Don't use the Association relation where another more specific relation can be used.
- If you cannot use a more specific relation, consider if objects are missing that would make modeling without the Association relation an option. Only fall back to the Association relation if a complete model would be too complex or detailed for the use you are going to make. In that case, develop a fixed pattern for the situation using the Association relation and reuse that exact pattern in comparable situations. In such cases, you should have at least once have modeled (analyzed) the complete situation in full without the Association relation as underlying foundation for the choices in your pattern. An example will be given later in the book when we will be talking about modeling ownership of applications and such.

- For the rest: only use the Association relation in its formal roles in the meta-model

Summarizing: use the Association relation only where it appears in the meta-model and in situations where doing a real model of what happens becomes too unwieldy.

### 8.4 Using properties too much

There is also an anti-pattern that has not so much to do with ArchiMate itself, but with tooling. Most tools let you add all kinds of information to an object or relation that is modeled. This is a good thing, for instance if the tool (almost all do) let you add the documentation for that object to that object.

Some tools also have the option to add explicit properties to an object type. This can even be in the form of free fields where you can add information. For instance, you could add the cost of an Infrastructure Service to the object and use that information to provide analysis of running costs of your Architecture.

Properties of objects and relations can be very useful, but they also have a danger: they are invisible most of the time. ArchiMate is a graphical grammar and those properties do not always show up when you want and they are not always available for the easy analyses you can make on the basis of a model. So, for instance, you might add the application's owner as a property to an Application Component, but that makes it impossible to connect that property to that same Actor that lives as a real Actor object in your model.

So, the bottom line is: only use properties for things you cannot model and even then be careful as they live outside of ArchiMate's grammatical structure.

The same limitations hold for labels. Labels do have a positive role to play in readability of course (see Section 6.6 "About labels" on page 43), but do not rely on them for analysis and structure.

## 9. An Example from the Real World

Ending this chapter I want to present you with a real-world example. This is about receiving files from an external data provider and using them. These files have to be received, moved to the right location in your landscape and then loaded in some application.

For this, the company employs a Managed File Transfer system, a Scheduler System and a human. The MFT System is programmed to receive certain files and depending on what it receives from whom, to move those files to a directory on a server in the landscape. There, the Scheduler watches for the file to arrive, and if it does, it starts a script that copies the file over to yet another location, using simple copy commands and shared file systems. After having done that, the Scheduler is done. The human (supposedly at some time of day) looks for the received file and loads it in the application by hand.

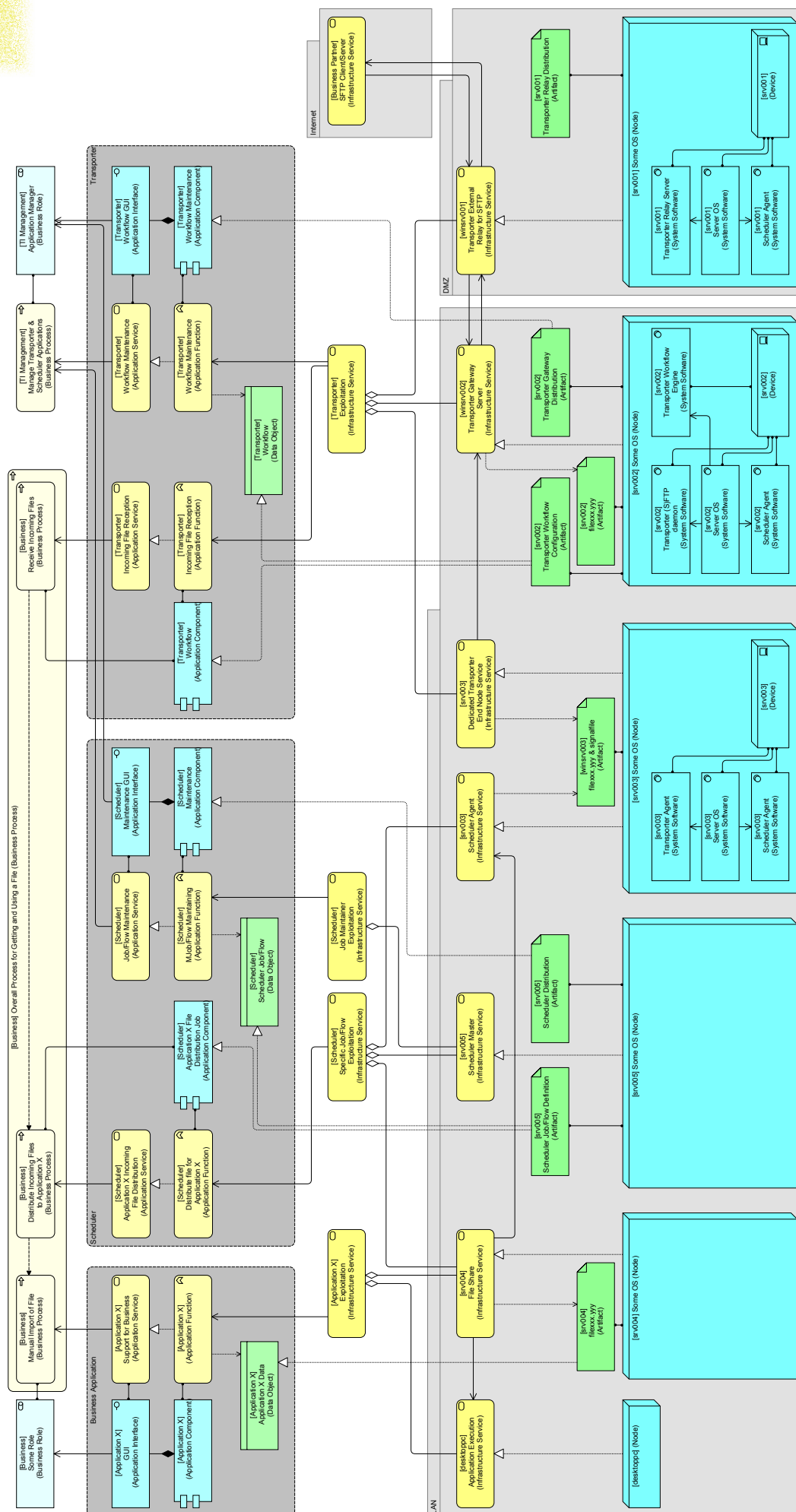
Now this situation is unnecessarily complex. After all, you could let the MFT system drop the file on the server where

the receiving system can read it. Assume that there are reasons not to do it that way.

In View 103 on page 66, this setup has been modeled. The MFT system works like this. In the Demilitarized Zone (DMZ) of our network, a 'Relay Server' (srv001) receives communication from the outside, e.g. an SFTP file transfer initiation. It checks if this party is allowed to and if so, patches the incoming communication through to a 'Gateway Server' (srv002) on the inside of our network. Here the actual receiving of the file happens and the file (Artifact filexxx.yyy) is stored locally. Here the MFT system runs a work flow that decides where the file has to go. In this case, it goes to srv003, a dedicated MFT transport end node: it receives all files together with signal files for the Scheduler. From here the Scheduler is taking over.

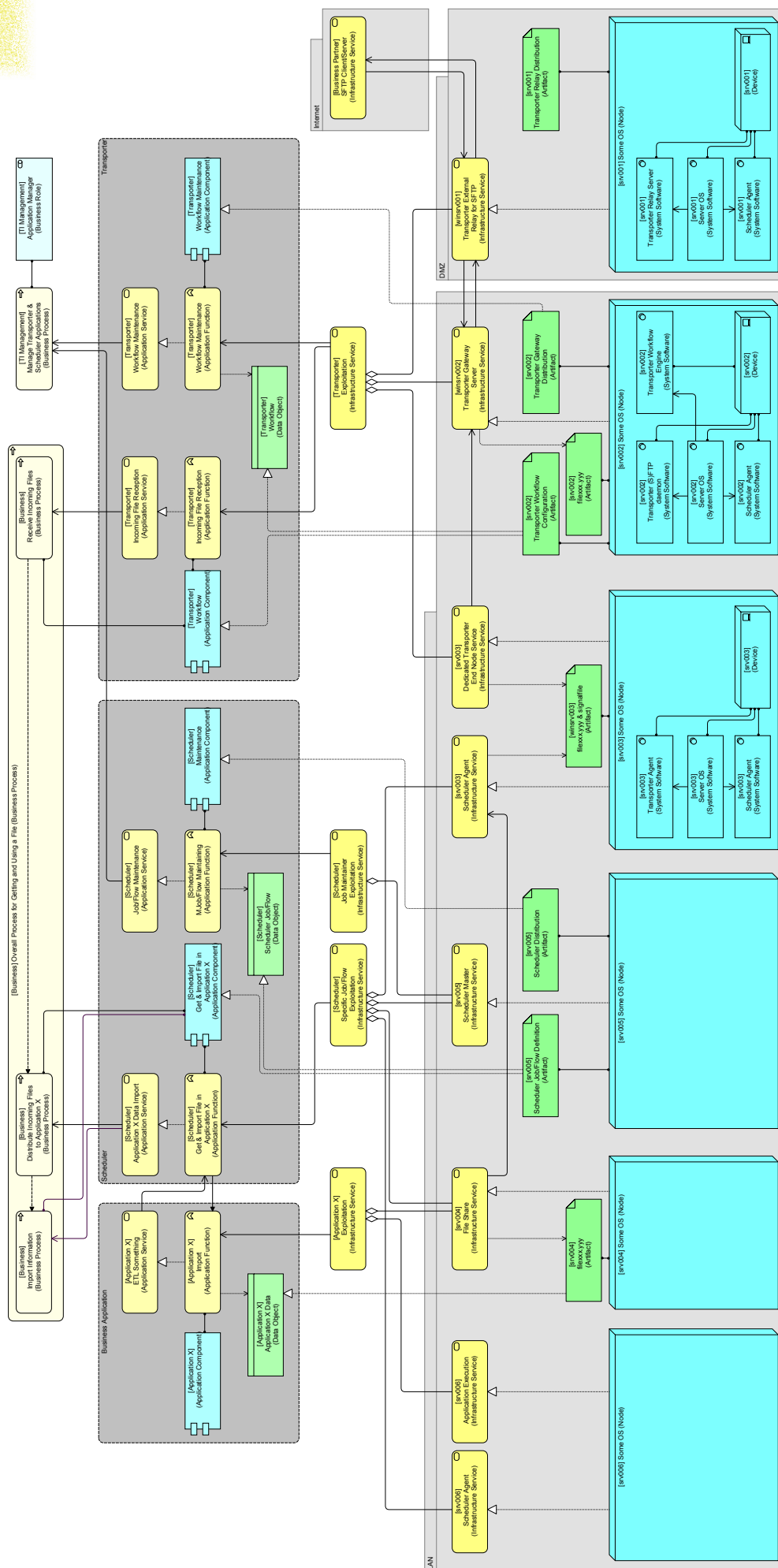
But before we go further I want to draw your attention to the other two Artifacts on srv002.





View 103. Modeling the Use of a Managed File Transfer and a Scheduler use for your business, a human user loads the retrieved data





View 105. Modeling the Use of a Managed File Transfer and a Scheduler use for your business, fully automated loading (cleaned up)



- One is the distribution of the MFT Gateway System. This Artifact Realizes the '[Transporter] Workflow Maintenance (Application Component)'. This application is used by IT Management to control the MFT System, create work flows for it etc.

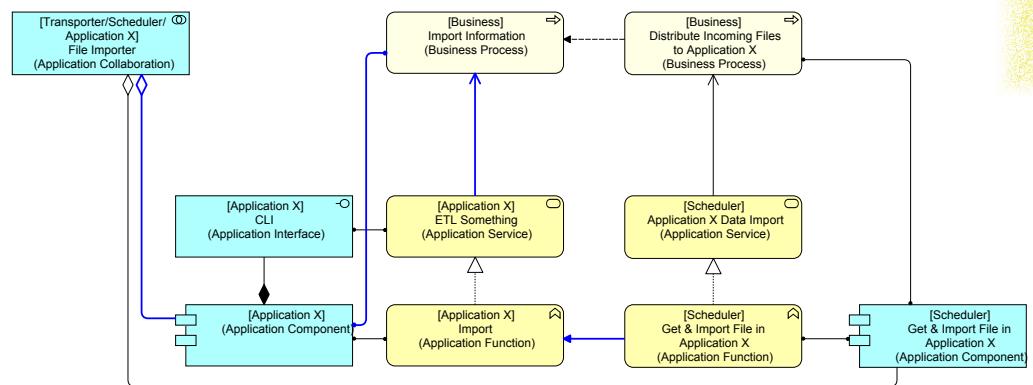
- The other is '[srv002] Transporter Workflow Configuration' (Artifact). This Artifact Realizes two different objects. One is the '[Transporter] Workflow (Data Object)', which is the *passive* object that is maintained by the 'Workflow Maintenance' application. The second is the '[Transporter] Workflow (Application Component)'. This object represents the workflow as an *actor* in our landscape which is responsible for automatically performing the 'Receive Incoming Files (Business Process)'. What you see here is both the architecture of *using* the workflow as well as *creating or maintaining* the workflow. In Section 12.3 "Secondary Architecture: Development" on page 84, this will be explained in detail.

The same primary (used by the business) and secondary (created/maintained by the developers) split is modeled again for the Scheduler.

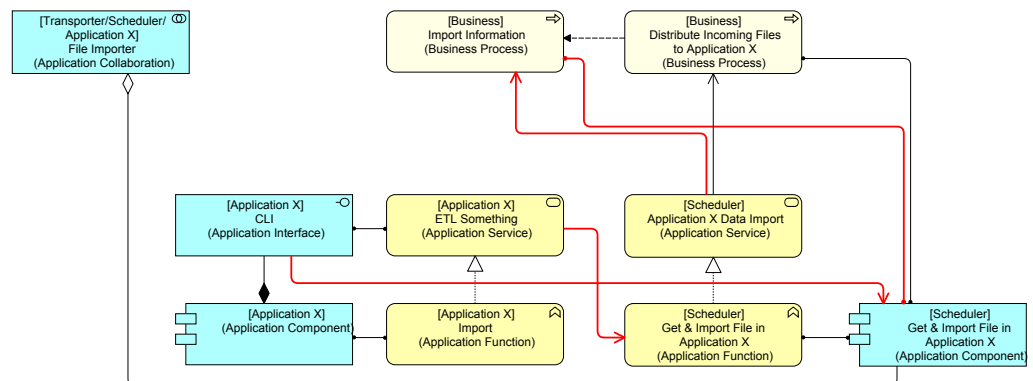
I also want to draw your attention to the way the file ends up on srv003 from srv002. On srv003 an MFT software agent is running that can receive files. This System Software Realizes the 'Dedicated Transporter End Node Service' that is used by the software on srv002. Following our deployment patterns, we do not model this in deep detail but we model this as a dependency between both Infrastructure Services.

The deep details of the Scheduler starting local scripts (jobs) on servers which then use mounted file systems to copy files are not modeled, there is no need. A shortcut is modeled, the File Share of srv004 is used by the Scheduler. Agent on srv003.

Have a look at the '[Scheduler] Specific Job/Flow Exploitation (Infrastructure Service)'. This is the 'exploitation' service we saw before (see View 86 "TI Building Blocks" on page 50). It is made up of the File Share where the file has to go, the Scheduler Master which is in control of the whole operation and the Scheduler Agent which is needed to do the actual copying. The Scheduler Agent therefore needs the File Share, and that is modeled by the Used-By between them.



View 106. Scheduling example: Scheduler triggers ETL



View 107. Scheduling example: Scheduler uses ETL

Finally, at the top of the view, you see the entire business process. Two subprocesses/steps are automatically executed by applications (the Transporter and the Scheduler systems) and one is performed by a human.

But that doesn't need to be the case. In many organizations such business processes may run fully automatic. This has been modeled in View 104 on page 67. This view is identical to the previous one, except in the upper-left corner. If the final step is not done by hand (by a human Business Role) but by a system, there are two ways to look at this (and probably more), shown in red and blue in View 104 on page 67.

- Following the blue relations and shown separately in View 106, we model that the loading is done by Application X, behavior that is *Triggered* by the Scheduler.
- Following the red relations and shown separately in View 107, we model that the loading is done by the Scheduler which *Uses* Application X.

Both are correct. But the one with the *Trigger* does not depend solely on structural relations, while the one with *Used-By* does. I find that somewhat cleaner.

There is another advantage for using the one with only structural relations. I generally do not model Interfaces, something that is more or less superfluous when you model the behavior well. (If someone disagrees with me, he or she can write his or her own book ;-). So, without those (and without the collaboration that I generally also do not use because in ArchiMate I cannot model a Collaboration of automated and non-automated actors, you get the result shown in View 105 on page 68. And for the entire set-up, only one relation is left.

# Advanced Patterns





# Advanced Patterns

## 10. Business Function and Business Process

### 10.1 On ArchiMate's divide between behaviour and structure

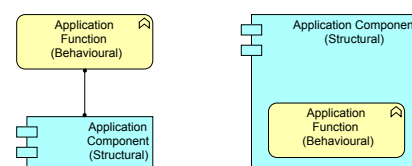
ArchiMate is divided into structure and behavior. Active structure objects (like Business Roles and Application Components) perform behavior (like Business Processes/Functions and Application Functions) and these behaviors act on passive structure (like Business Objects and Data Objects). This division is a fundamental property of the language, but it is slightly different than other ways to look at architecture (e.g. traditional view of software architecture or business architecture) and that does sometimes lead to discussion.

What I have noticed is that especially ArchiMate's Business Function object is problematic in this sense. It has been customary in some circles to see a Business Function as a somewhat visible 'part of the organization' that performs a certain function in that organization. And that description shows already where the problem lies. In non-ArchiMate ('old school') business architecture speech, a business function often *performs* something and both there and in ArchiMate terms, something that *performs* behavior is an *active* structure, while ArchiMate's Business Function in reality is not an active structure object, but a *behavioral* object that *is* performed.

Application Function also suffers from this problem. In the field of Software Engineering, a function is both a piece of code (which is structural) as well as the behavior of that piece of code. Historically, there have not been two terms in software engineering to separate that structure and its behavior (nor in the related subject of mathematics, where formula and behavior of a function are one and the same). Also, coming from software engineering, in UML the division between structure and behavior is different from that in ArchiMate. In UML, we have a set of objects, some of which may play both a structural and a behavioral role, depending on the type of diagram they are in. Take for instance an Object Diagram in UML, which is structural, and a Sequence Diagram, which is behavioral, and both may use the same objects in either a structural or a behavioral role. In other words: UML allows objects that can be both structural and behavioral, depending on the views they appear in (structure type of view or behavior

type of view). This follows the paradigm of Object-Orientation in software engineering, where objects are structural but encapsulate behavior (and some data even). In ArchiMate, there is but one type of view (even if you are free to restrict yourself to certain object and relation types in different viewpoints) which combines structure and behavior, but the objects themselves are separated in behavioral and structural (and passive). Coming from one world, the other doesn't fully match. (UML is big, complex and mostly directed at the world of object oriented software engineering, it is not really a good placeholder in this section for software engineering in general and much more can be said about it, but I thought the juxtaposition was nice).

So, both in business descriptions and in software engineering, many people see structure as something that 'encapsulates behavior' all by themselves. In ArchiMate, it is possible to suggest a structure that encapsulates behavior as can be seen in View 108:



**View 108.** Nesting a behavioral object inside an active object to suggest encapsulation

On the left you see the un-nested model, with an Assignment relation between the Application Component and the behavior of that component: the Application Function. On the right, a visualization suggesting encapsulation using ArchiMate's Nesting relation, here used for Assigned-To (nesting is one of those parts where ArchiMate is pretty unclear as it can mean (a mix of) three different relation types. I personally almost exclusively use Nesting for Composition between objects and, with a bit of reluctance, Aggregation). Note: though this nesting suggests encapsulation, it does not *mean* encapsulation. "Encapsulation of behavior equals structure" does not exist in ArchiMate.



Back to the business layer. ArchiMate's definition for a Business Process is:

*A business process is defined as a behavior element that groups behavior based on an ordering of activities. It is intended to produce a defined set of products or business services."*

ArchiMate's definition of a Business Function is:

*A business function is defined as a behavior element that groups behavior based on a chosen set of criteria (typically required business resources and/or competences).*

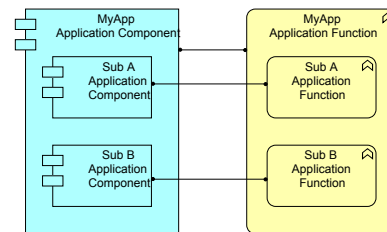
In summary, both are behavioral and both group the *same* activities (but from a different perspective). ArchiMate 2.0 makes also clear in its explanation that both process and function are the behavior of a single role and that if you want to model behavior of multiple roles, you should use a Business Interaction, which is the behavior of a Business Collaboration.

So, a Business Process groups behavior based on what it *produces*. It is a grouping based on an 'outside' parameter: the *result* of the activities. A Business Process is therefore an 'outside-in' grouping of behavior. A Business Function groups activities based on an 'inside'-parameter: what resources and capabilities it needs. A Business Function is therefore an 'inside-out' grouping of behavior, whereas a Business Process (a grouping based on what is visible on the outside, like a service or product) is an outside-in grouping of (potentially the same) behavior.

Both Business Process and Business Function both are *behavior* in ArchiMate. For Business Process, that is not surprising for most of us. But for Business Function, that differs from other approaches where a function may be considered structural. In ArchiMate, Business Function is *purely* behavioral, it is itself behavior that may be a grouping of (sub)behavior. So, where is the structure that performs it? Well, according to the definition, a (single) Business Role (in practice often a role fulfilled by a department). So, where a business function in other approaches may be a structure that encapsulates behavior and thus be structure and behavior in one, in ArchiMate, a Business Function is pure behavior and it is performed by a different, structural object: a Business Role. (ArchiMate 2.0 repaired the somewhat confusing definition of a Business Role in ArchiMate 1.0 as (named) *behavior*).

And what about grouping/encapsulating of Application Component and Application Function? Same here and even nicer. Because at the application level, ArchiMate kind of sees the same sort of 'encapsulation' on both sides. Application Function may be an encapsulation (by means of a composition) of other Application (sub)Functions and Application Component may be an encapsulation (by means of a composition) of other Application (sub)Components. Between the Application (sub)Functions and Ap-

plication (sub)Components is a one-on-one Performed-By relation. As can be seen here in nested form in View 109:



**View 109.** Encapsulation of Application Sub-Components and Application Sub-Functions

Seeing the composition of Application Functions on the right easily makes one use the word 'structure'. And indeed, in a sense, there is a 'structure' in the Application Function, as it is a composition of Application (sub) Functions. But though there is a 'structure', that does not mean the object itself is 'structural' in ArchiMate's sense. Paraphrasing Uncle Ludwig: watch out for 'bewitchment by language'. The fact that you are using (almost) the same word ('structure') does not automatically make it the same meaning.

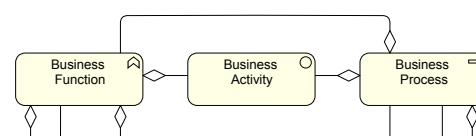
## 10.2 Business Function or Business Process?

So, a rather tricky part of ArchiMate is the difference between Business Process and Business Function. Both stand for behavior at the Business Level. Both generally encapsulate in the end the same activities. Choosing between a Business Function and a Business Process is sometimes difficult. It is like the eternal question about the chicken and the egg. Note: there is a large part of the architectural community that thinks it is not difficult at all. For them, a process is a chain of functions. But the story is more complex.

**Question 5.** Now that I'm mentioning it: Which was first, the chicken or the egg?

**Answer 5.** The egg. The first chicken egg was a mutation laid by a pre-chicken from a mutated seed or egg cell. A chicken is the means by which an egg makes another egg.

ArchiMate in its prenatal form used to have a concept called Business Activity. This would stand for a lowest level, indivisible, piece of business behavior. It was not to have any constituent parts. Business Function and Business Process would then be an Aggregation of business behavior objects, either Business Function, Business Process or Business Activity. In an 'unfolded' picture that lacks the abstract 'Business Behavior' concept, it looks like View 110:



**View 110.** Pre-ArchiMate 1.0 relations between Business Process, Business Function and Business Activity

They dropped activity in the final 1.0 spec. I do not know why exactly, but I can imagine a couple of reasons:

- It is difficult to decide when something becomes indivisible;
- It is difficult to come up with a clear distinct definition of Business Activity that is neither inside-out or outside-in.

My initial approach to combining the different roles of Business Function and Business Process in our ArchiMate models was close to this original, but with a small twist. In our company it was still customary to think of Business Processes as being a sort of 'chain' of Business Functions (as is often the way architects look at the division) and in ArchiMate this can become a Business Process that has aggregated Business Function children. I proceeded to make that view symmetric in that one could also say that a Business Function aggregates the Business Processes it contributes to. This can be seen in View 111.



**View 111.** A Business Process Aggregates Business Functions and a Business Function Aggregates Business Processes

(I had removed activity as it is not part of ArchiMate 1.0)

In this way, I took the classic approach and made it nicely symmetric. Both are then just overlapping (not necessarily orthogonal) views of the same business behavior. Say, a wave/particle view of something that is essentially the same thing, one 'measured' from the aspect of what it produces (process, outside-in) and one of what it requires (function, inside-out). I can personally live with that Quantum-Mechanical-like ambiguity, but it is not to everybody's liking. Therefore, I dropped that way for another, because we actually can be more precise in our thinking using ArchiMate's core relations from the metamodel as guidance.

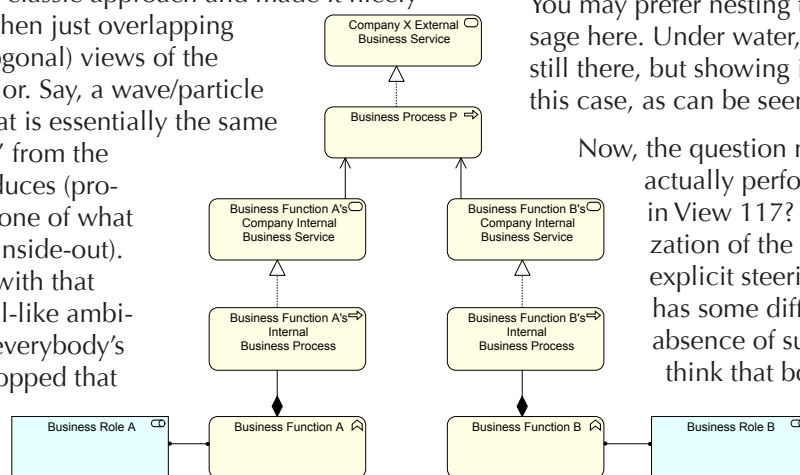
Suppose we do indeed see process as a kind of 'chaining' of functions, what are we actually saying, ArchiMate-wise? Chaining is not an ArchiMate relation, after all. Luckily, ArchiMate has a good relation for this: Used-By.

So, what we can say is that a Business Process *uses* the Business Functions in some way (instead of both being aggregates of each other). But, in ArchiMate, how does a process use a function? Well, a Business Process can use a Business Service, which in its turn is provided by the Business Function.

But if you want to model the behaviour that *Realizes* a Business Service, the preferred option is a Business Process and not a Business Function. After all a Business Process is 'intended to produce a service'. So we must ask ourselves: if a Business Function *Realizes* a Business Service what *process* actually realizes that service? Well, that must then be an *internal* process of the Business Function. The result is something like View 116. Here we see a Business Process that uses ('chains') two different Business Functions to realize a Business Service. The roles are assigned to the functions, that according to ArchiMate should be performed by a single role. The internal processes of the functions have here been modeled as composites of the function, not aggregates. That is logical, because the *internal* process of a business function ceases to exist if you delete the business function.

This pattern can of course be repeated infinitely, i.e. the internal process uses multiple internal sub-functions and their internal sub-processes. I think you should be careful with that, as it fragments your model. It depends on the future questions you want to answer with the help of the model if those details are really useful.

You may prefer nesting to strengthen the visual message here. Under water, the composition relation is still there, but showing it embedded is kind of nice in this case, as can be seen in View 117.

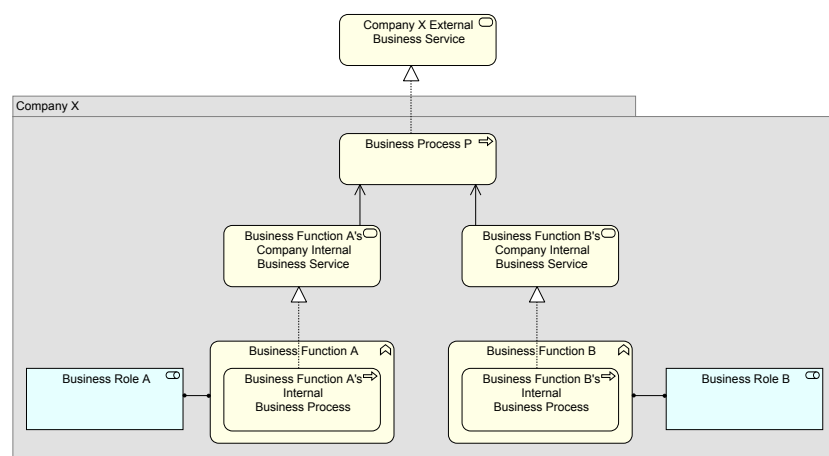


**View 116.** A Business Function's internal process Realizes a Business Service

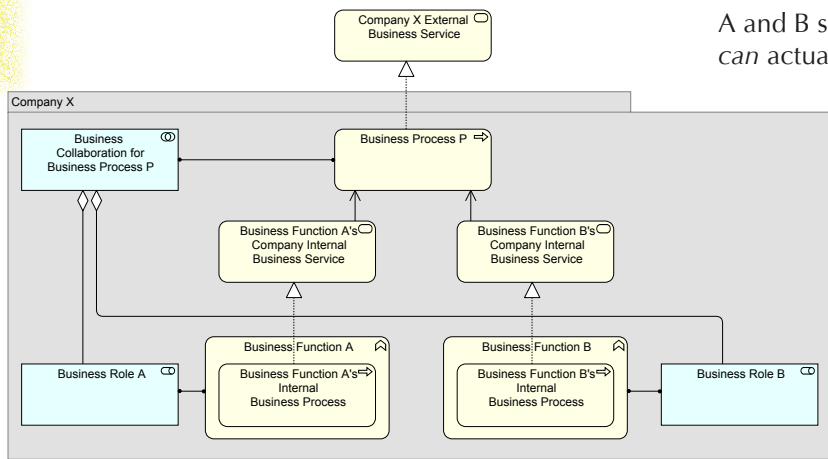
Now, the question must be asked: which role(s) actually perform(s) "Business Process P" in View 117? This depends on the organization of the company. Maybe there is an explicit steering role for that process (which has some difficulties if you go into it). In absence of such a steering role we could think that both Role A and Role B are

assigned to Process P. But the (ArchiMate) derived relations between said roles and process from the model above are not Assigned-To but *Used-By* (performs+composite+realizes+used-by = used-by), so it would be confusing to have both Assigned-To and Used-By from roles B and A to process P. Therefore, we need another separate role to assign to process P.

A solution is to use a Business Collaboration consisting of the roles in question here to assign to the process. This is not that far from

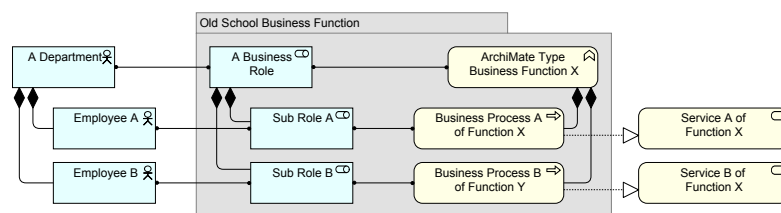


**View 117.** A Business Function's internal process Realizes the Business Function's service, nested view

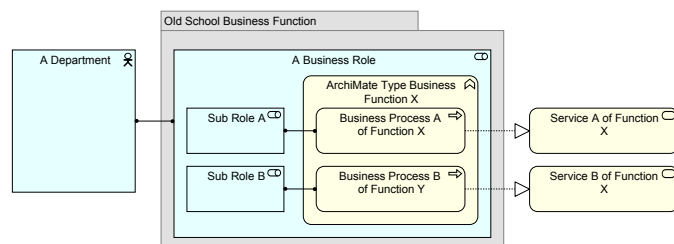


**View 118.** A Business Process using Business Functions is performed by a (loose) Business Collaboration

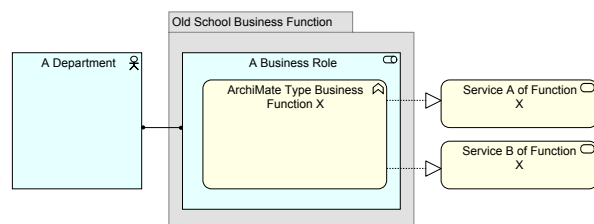
what in reality often happens. In many companies we will see some sort of very loose and thin 'collaboration'. Payments just pays what Claim Handling has approved and they only communicate when it is about the process itself or when something is amiss (which in turn is often all that they collaborate about). There is no separate role making sure that everything happens. Yes, there are (as oversight) process owner roles for the internal processes of function



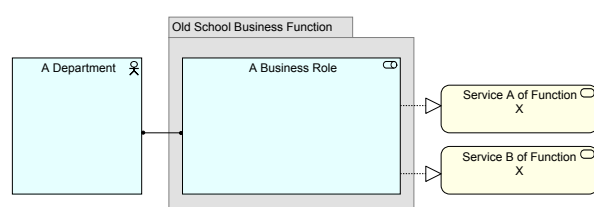
**View 119.** An ArchiMate Business Function, with two internal processes, roles and services realized



**View 120.** An ArchiMate Business Function, with two internal processes (nested), roles and services realized



**View 121.** An ArchiMate Business Function realizing two services with internal processes omitted from View 120



**View 122.** A Good-Old-Fashioned-Business-Function becomes a Business Role in ArchiMate, further simplified from View 121

A and B so their internal process is such that collaboration *can* actually happen. But these, as we will see in Section 12 "Secondary and Tertiary Architecture" on page 84, are quite different roles than those that are assigned to performing the function. Anyway, an example of loose/thin collaboration would be something like View 118 (note: this is sloppy ArchiMate as the collaboration is assigned to a process and not to an interaction as is proper).

Depending on how the operating model of your organization actually is, a different picture may emerge of course.

If instead of using an 'end-to-end' process that uses functions, but we model the End-to-end process as a Business Interaction and notice that it can Realize a Business Service, you can wonder about the internal setup of that interaction. ArchiMate just says the interaction is performed by the collaboration and leaves it at that. But if you look deeper, what happens when different roles collaborate?

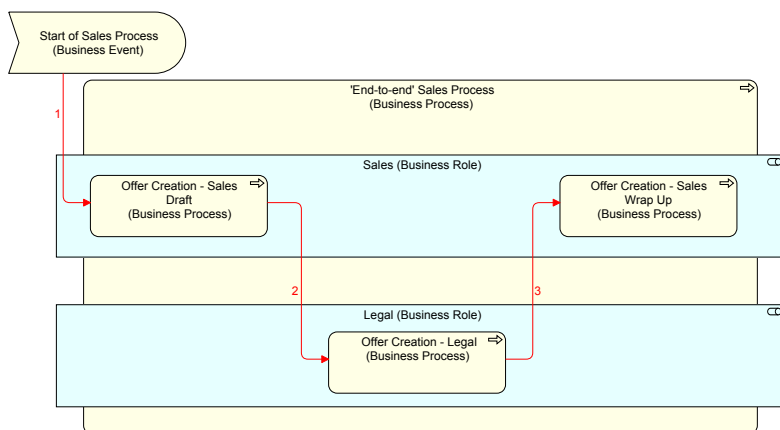
The first thing is that if different roles collaborate, it is their separate behaviors in the collaboration that *together* creates the interaction. These separate behaviors will somehow be recognizable as such also, or you will not be able to define the interaction in your business manuals. If two Business Functions together somehow create the Business Interaction, it is either an 'interaction of processes' or 'an interaction of functions'. Since an interaction is directed at *realizing* something (a service for instance) it is best to use processes. We'll show an example below when we look at the relation between ArchiMate Enterprise Architecture modeling and Business Process Modeling.

If we take the approach in View 118 and map it to the application layer, we would get the situation that we might say that an Application Function has its own internal Application Process, a concept that does not exist in ArchiMate. Should it? The 'Application Process' realizes Application Service and Application Function is grouped based on Data Objects and Infrastructure Services required? Should Application Process stand for the algorithm that produces the result? It might give a nice starting point when you think about business rule engines. Still, as we are not discussing changing the language here, but describing how to use it as it is, we will not go into that here. Besides, at the Application Layer, it is pretty difficult to split the idea of a function and a process. However you model it, one can be turned perfectly into the other.

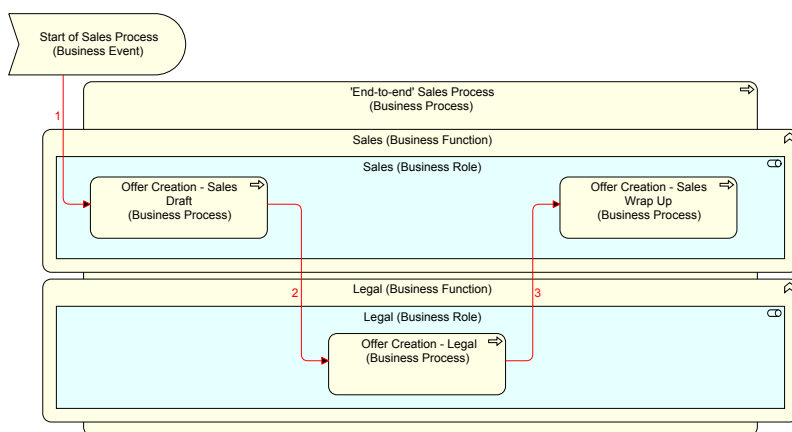
## 10.3 Good Old Fashioned Business Function

In Section 10.1 on page 71, I discussed the structural/behavioral divide in ArchiMate and how it differs from the 'old school' Business Function (something I call the GOFBF or Good-Old Fashioned Business Function in honor of Hubert Dreyfus, the philosopher who wrote *What Computer's (Still) Can't Do*, but I digress), the 'structural' function that encapsulates 'behavior'. Here, I'll show a way to model this in ArchiMate.

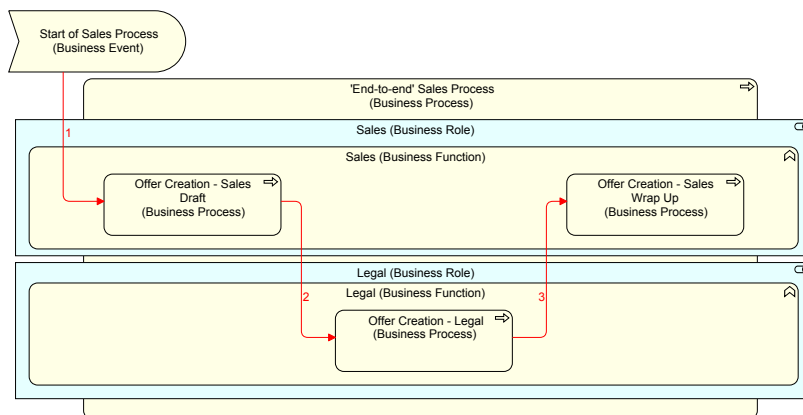




**View 123.** How the process modeler looks at the 'end-to-end' process in a 'swim lanes' kind of view



**View 124.** Business Functions added to the process modeler's View 123, the GOFBF way



**View 125.** The process modeler's View 123, with functions added the (kind of) ArchiMate way

The full world of a Business Function delivering services to the organization can be modeled like View 119 on page 74. The gray box contains the 'old school' business function. The structure (role with sub roles) and the behavior (function and internal processes). That doesn't look like 'encapsulation' yet, but using ArchiMate's Nesting visualization, we can make a view that suggests the encapsulation as in View 120. It is a bit nasty, this one, because the relation between 'A Business Role' and 'ArchiMate Type Business Function X' is Assigned-To, something not always expected of an ArchiMate Nesting.

We can simplify this picture further as can be seen in View 121 (note, all these views come from the same model, they are just presented in different ways to convey a different message). The Realization relations from 'ArchiMate

Type Business Function X' to 'Service A of Function X' and 'Service B of Function X' are derived from Composition (from function to process) and Realization (from process to service).

We can remove ArchiMate's Business Function from the view and summarize one level further to get one single object in the gray grouping of View 122. And there you have it: a GOFBF viewed as a structure encapsulating behavior. Conclusion: If you want to model a GOFBF, you should use Business Role in ArchiMate. Incidentally, ArchiMate says that a Business Function or a Business Process must be seen as the *internal* behavior of a Business Role. So, there you have it, it was there already from the start...

## 10.4 The 'End-to-end' Business Process

In Business Process Modeling many model without functions at all. They model so-called 'end-to-end' processes that consist of steps and subprocesses that are performed by different roles. This is different from ArchiMate and that has to do with ArchiMate's split of active and behavioral objects.

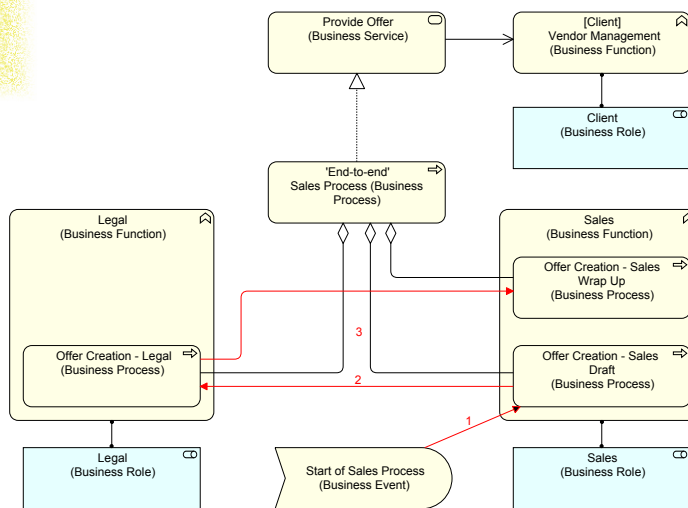
To explain that, I am starting with a view that depicts the thinking of a process modeler (View 123). The process modeler looks at the end-to-end process as a series of subprocesses or process steps that are performed by different roles or actors in the organization. Note: View 123 suggests something not exactly ArchiMate. For instance, though the subprocesses are nested in the roles (thus modeling swim lanes) the subprocesses are *also* embedded in the end-to-end process. In my model, the subprocesses are aggregate children of the end-to-end process, but in the view, the end-to-end process is positioned *graphically* behind the role 'swim lanes'. (I can also nest the roles inside the end-to-end process in my tool, even if they stick out.)

The first thing you may wonder about is the question where the Business Functions are. And indeed, most of the time I have talked with business process modelers they tend to ignore the functional division of the organization,

other than the roles. In fact, this is one of the ways the GOFBF-thinking manifests itself. The process modelers do not make the separation of role and function too clearly. Process modelers tend to see a business function as an 'actor' of sorts in the GOFBF way. The Process Modeler's 'business function' X *performs* Y, whereas in ArchiMate the business function is just another categorization of behavior of actors (X is *performed by* a role).

We can add the Business Functions for which the subprocesses are internal processes. This can be seen in View 124 and View 125.

If you think in the GOFBF-way, where 'functions' are structural with roles inside them, you get View 124 (remember, this all is not really proper ArchiMate) In ArchiMate, the



**View 126.** The end-to-end process as a chain of processes, without using collaboration and interaction

nesting of roles and functions toggles: a function may be nested *within* a role (thus depicting the Assigned-To relation to the roles' behavior). This can be seen in View 125.

Note, this is borderline clean modeling in ArchiMate. The Nestings of processes inside functions depict Compositions, the Nestings of functions inside roles depict Assignment and the Aggregation relation between the end-to-end process and the subprocesses is not really shown.

We can model the end-to-end process in ArchiMate also according to the patterns used in Section 10.2 "Business Function or Business Process?" on page 72. View 126 on page 76 is in fact equivalent (with the exception of the added client) to View 125, though some relations are now shown explicitly but instead in the form of Nesting.

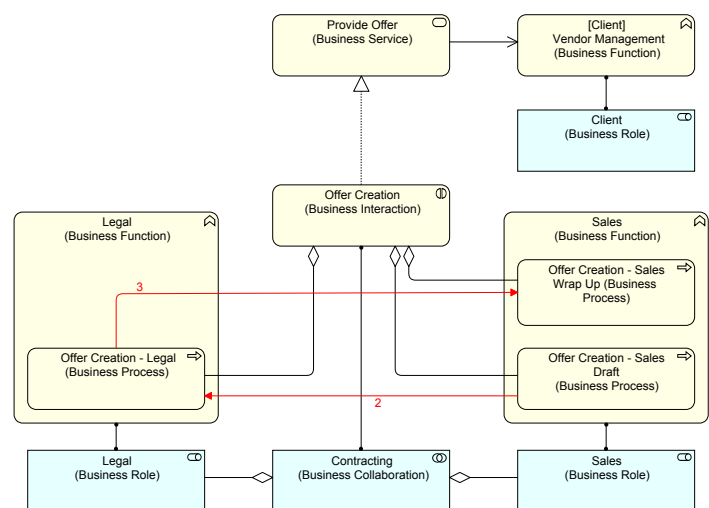
We have another option. We can use a Business Interaction to model the 'end-to-end' process. After all, the 'end-to-end' process is performed by a collaboration of the roles that perform the processes that 'make up' the interaction. This can be seen in View 127 on page 76.

If we follow strictly what ArchiMate says about interactions being just the behavior of a collaboration (ArchiMate, remember, says nothing about the interaction being an Aggregate collection of processes like it says the collaboration is an Aggregate of roles) we get View 128. The problem here is that you lose the direct relation between the 'end-to-end' process and its constituent parts. Since ArchiMate does not forbid the approach of View 126 or View 127, these can be used. The advantage of using the Business Interaction as in View 127 is that you can see for every 'end-to-end' process exactly which roles are required where and this remains the case when you start to get more complex models with sub-roles, etc.

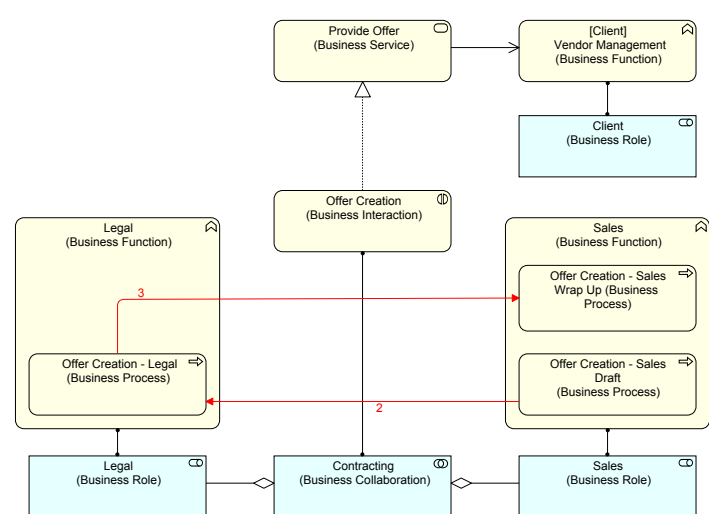
## 10.5 Business Proces Modeling versus Business Layer Architecture

There is one aspect with modeling where I have encountered a fundamental different outlook between some Business Process Modelers and ArchiMate. That difference is their attitude towards automation.

When these process modelers look at the position of automation in the business processes, they tend to focus solely



**View 127.** End-to-end process modeled as Business Interaction (with added Aggregate relations to the subprocesses)



**View 128.** The end-to-end process modeled as a Business Interaction, strictly following the description of Business Interaction in the ArchiMate 2.0 specification

of what they can describe of the human part of the process. The automation part has to be described elsewhere (in the documentation of the systems). This means that if a process is automated, in they eyes of these business process modelers it disappears from their radar. It has become automation, not process. ArchiMate, however, looks at it differently. Even if a business process is automated, there still is a business process.

Personally, I think ArchiMate is superior. The way to look at automation as something that is 'used-by' human processes is only part of the story. And more and more, automation runs processes entirely by itself. For most companies, fully automating certain processes and making the business architecture 'STP' (Straight-Through Processing) and 'Exception Based' is the new norm. But process modelers are not often well aware of Enterprise Architecture's integral view of the landscape and look at it 'the old way'. The best way to confront the issue is to ask what would happen in their models if a process was fully automated. Where in that case do we find the process?

Note that the process modeler's view is easier on the business user who generally does not think in abstractions like a 'Report Definitions (Business Object)' but thinks of the 'reportdefinition.xls' file.

# 11. How ArchiMate Helps Choosing Your Business Functions

## 11.1 Dividing your enterprise's behavior into Business Functions

Having discussed how to combine Business Functions and Business Processes, let's look at an actual discussion I once had with other architects. For this, I first need to make a short and simplified description of what was under discussion.

In Asset Management, portfolio managers (or fund managers, etc.) manage assets. They have a portfolio of holdings, stocks, bonds, or maybe ownership of shopping malls etc. They manage these holdings generally along three aspects:

- **Performance:** what does this investment actually deliver in profits?
- **Risk:** what is the risk associated with this investment? How uncertain are the profits or the value of this investment?
- **Compliance:** are we following the restrictions set upon us? An example may be that the client has told you not to invest in weapons. Or the organization has generic risk-minimizing rules like that you are not allowed to invest in certain countries, or trade with certain counterparties, or invest in certain types of instruments (like junk bonds).

Asset Managers try to maximize financial performance while minimizing financial risk. In the meantime they all work under the assumption that there is generally an inverse relation between the two, so if there are more risks, they will want to see more performance. E.g.: riskier bonds have to pay out more interest, safer bonds deliver less. They try to mix assets with different risk profiles as these combine generally into a lower risk of the portfolio. They also try to minimize their own cost and they have to comply to the compliancy rules.

Now, the process (simplified) is as follows. Each day, the portfolio manager inspects his portfolio and the news in the world. When inspecting his portfolio, he tries to assess future performance, he assesses risk (often based on all sorts of statistical calculations) and then he may decide to buy or sell something. When he has decided (say, buy \$100 million worth of Zimbabwean diamond mining

stock), there will be a phase of so called 'ex ante' (Latin for 'before') compliance checking. There are many kinds of checks, based on rules set by a different department, the 'investment control' department. Generally, this department monitors performance, risk and compliance and sets the rules for compliance. One of those rules may be that there is a limited list of countries the portfolio manager is allowed to invest in, and Zimbabwe is not on that list. The check bears this out and such the order is not passed on to the trading desk.

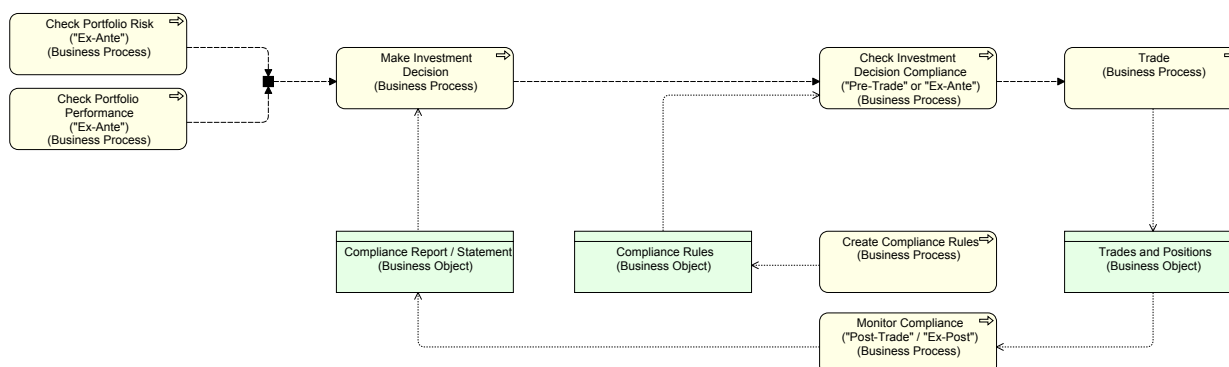
Some asset managers let the portfolio managers or traders do their own compliance checking on the basis of the instructions, like the 'allowed country list'. Some separate these duties away from the portfolio manager to make it less likely a portfolio manager enters into a deal that has to be rolled back. Of course, the risk difference between the two are limited as any compliance break will turn up when the 'ex post' (Latin for 'after') checking at the end of the day is performed by investment control.

Now, here is the discussion: When architects discuss the business architecture of this asset manager and they want to divide the activities up into business functions, the question arises: is 'ex ante' (what happens before the deal) compliance checking part of the 'portfolio management' business function or part of the 'compliance control' business function, which also has 'ex post' (after the deal) checking? Two things get easily mixed up here: functions and processes on the one hand and roles and actors on the other. But let's start with an initial view of the processes and objects involved as seen in View 129. 'Investing' processes are

- Check the portfolio against performance and risk parameters;
- Make an investment decision;
- Check the decision against compliance rules;
- Make the deal (trade).

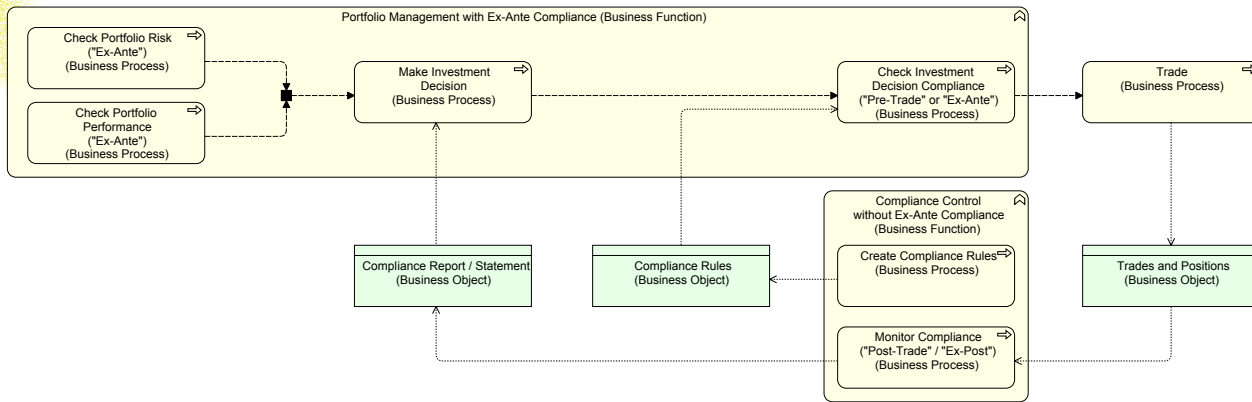
The compliance processes (we have limited ourselves to compliance here, you have also processes that measure risk and performance) are:

- Create the compliance rules and objects;

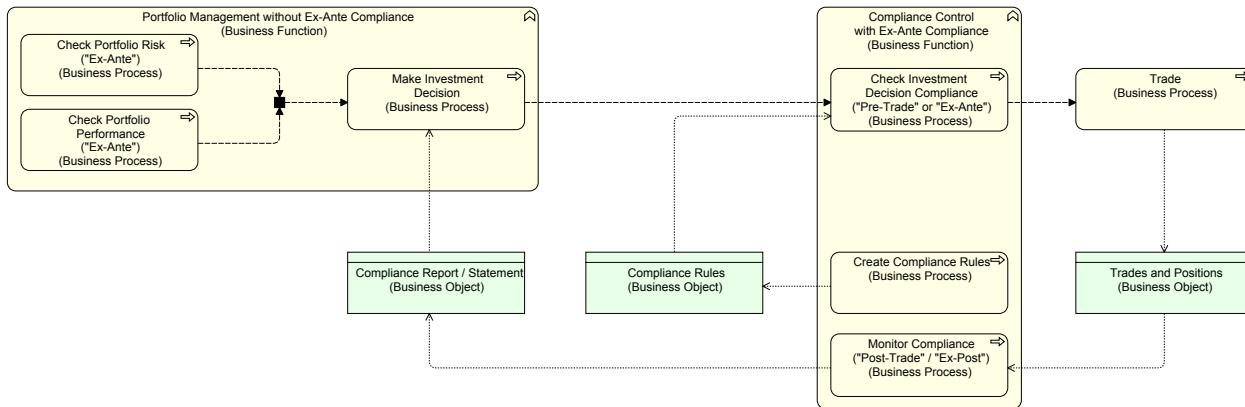


View 129. Simplified Portfolio Management without Business Functions





**View 130.** *Simplified Portfolio Management with Ex-Ante Compliance as part of Portfolio Management function*

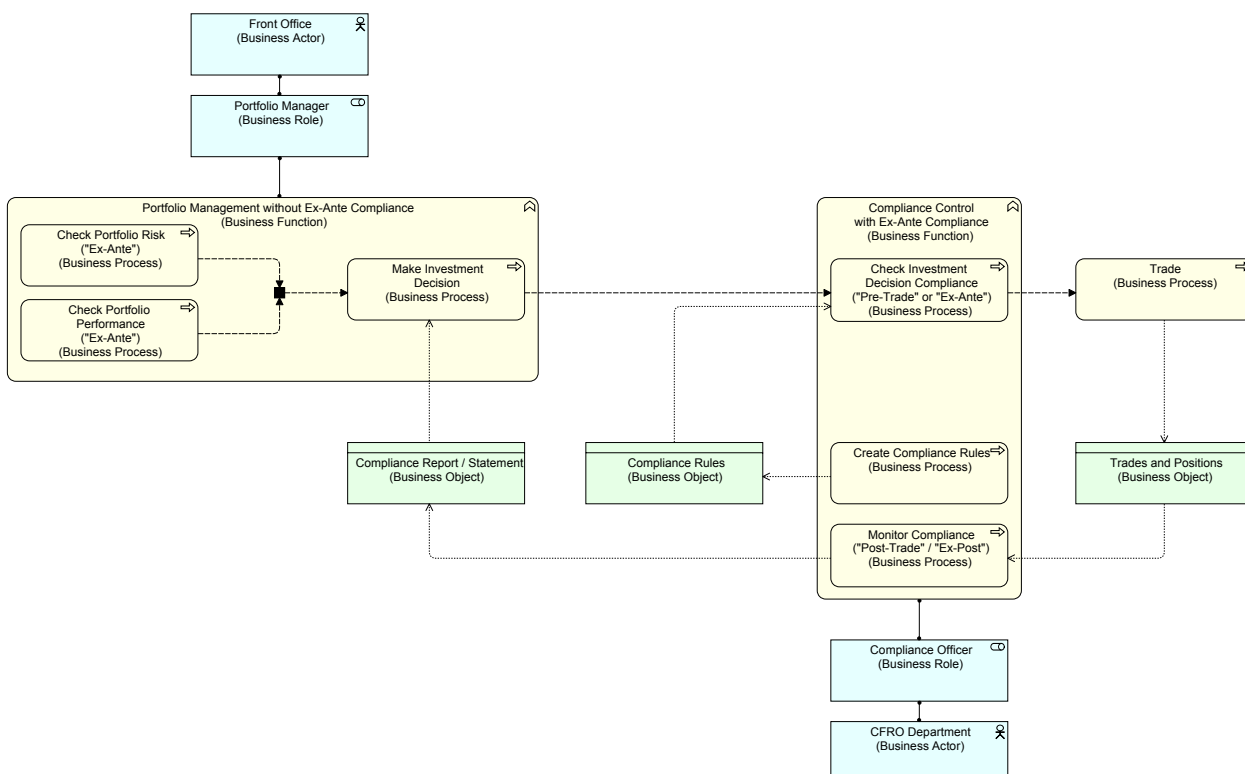


**View 131.** *Simplified Portfolio Management with Ex-Ante Compliance as part of Compliance Control function*

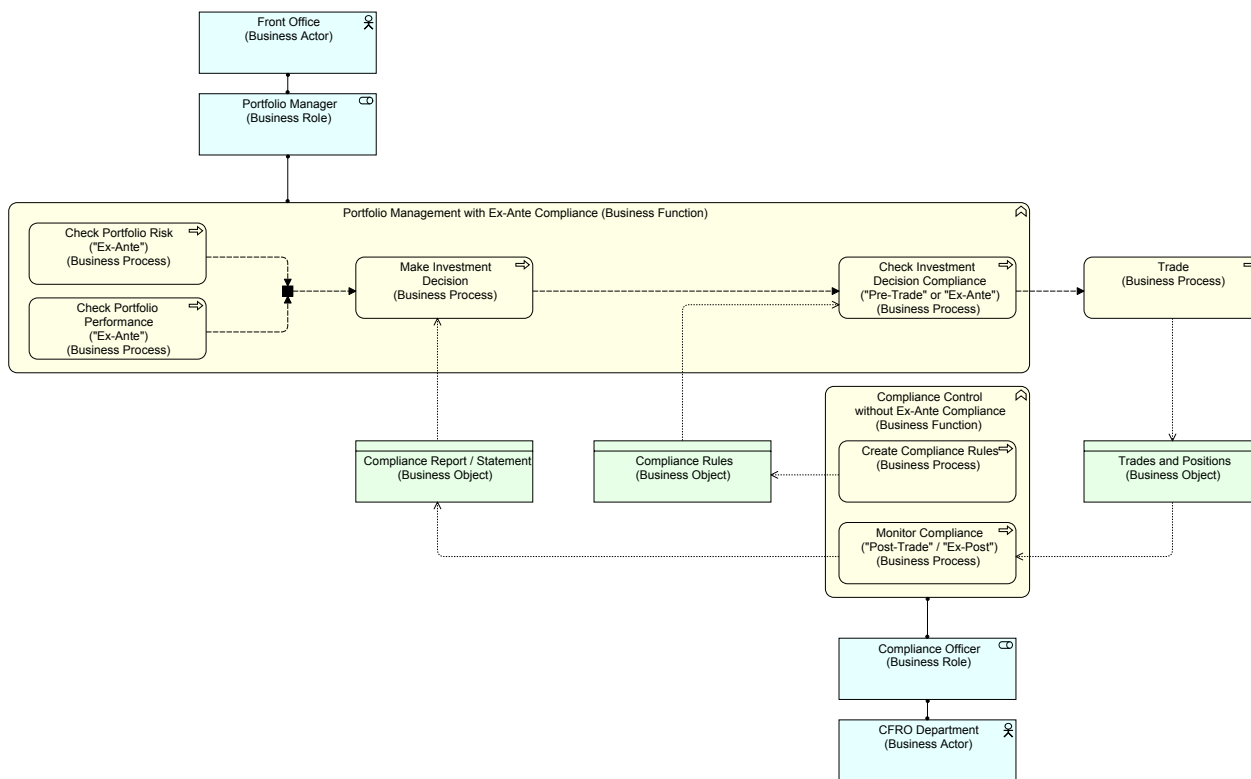
- Monitor the trades and positions against compliance rules;
- Check the decision against compliance rules.

The latter can be seen as part of both ‘investing’ and ‘controlling’, it seems, and in fact organizations disagree on this point. Some see it as ‘investing’, just as checking performance and risk are part of the normal activities of a

portfolio manager (View 130). Others see it as part of all of the ‘compliance’ activities that are performed by a different group than the portfolio managers, say the investment control department (View 131). This difference is in fact based on difference in actors more than in other differences. In ArchiMate that is actually a pretty decent reason for separating activities into different Business Functions as



**View 132.** *Simplified Portfolio Management with Ex-Ante Compliance as part of the Compliance Control Function*



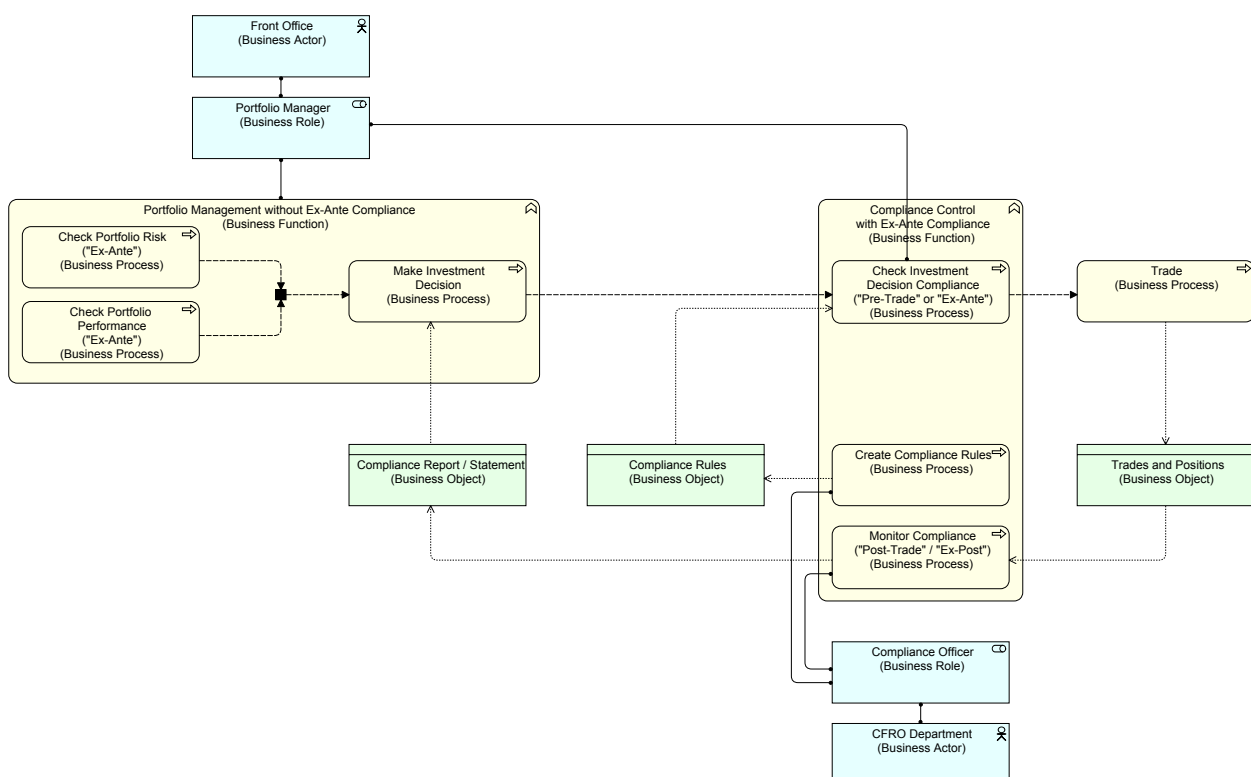
**View 133.** *Simplified Portfolio Management with Ex-Ante Compliance as part of the Portfolio Management function*

functions are groupings based on shared resources, skills, etc., which are performed by a single role.

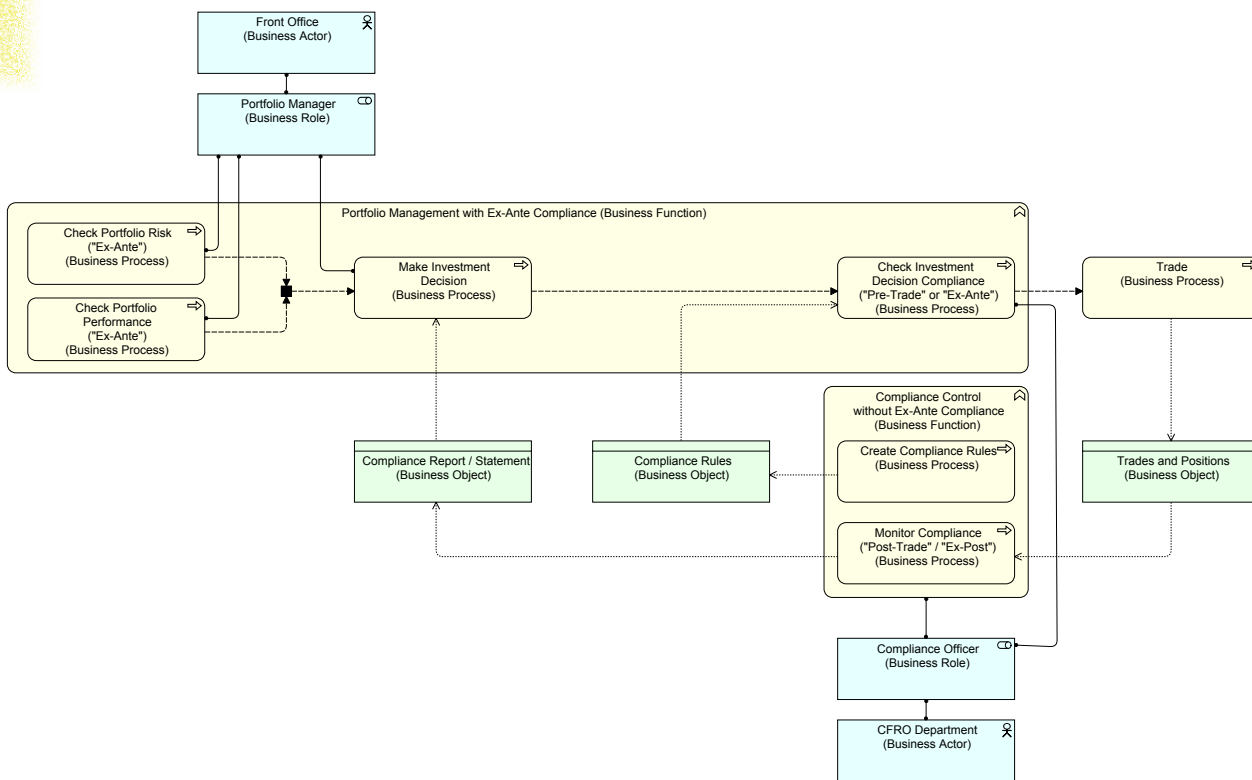
So far, so good. Let's add those roles and actors to the main Business Functions in both solutions. The Portfolio Manager role performs the Portfolio Management Business Function and the Compliance Officer role performs the Compliance Control Business Function. When the 'Ex-Ante Compliance' is seen as part of 'controlling', performed by a compliance officer, it looks like View 132. When 'Ex

Ante Compliance' is seen as part of 'investing', it looks like View 133. Both are pretty clean views.

But 'being performed by a compliance officer' is just *one* possible reason to separate activities into a Business Function. There are many more, other resources, for instance. And suppose we would have the following situation: the portfolio manager does the 'Ex Ante Compliance' check, but 'Ex Ante' compliance checking is still seen as part of the 'compliance' function, e.g. because it is supported by the same IT services as the other processes that are part



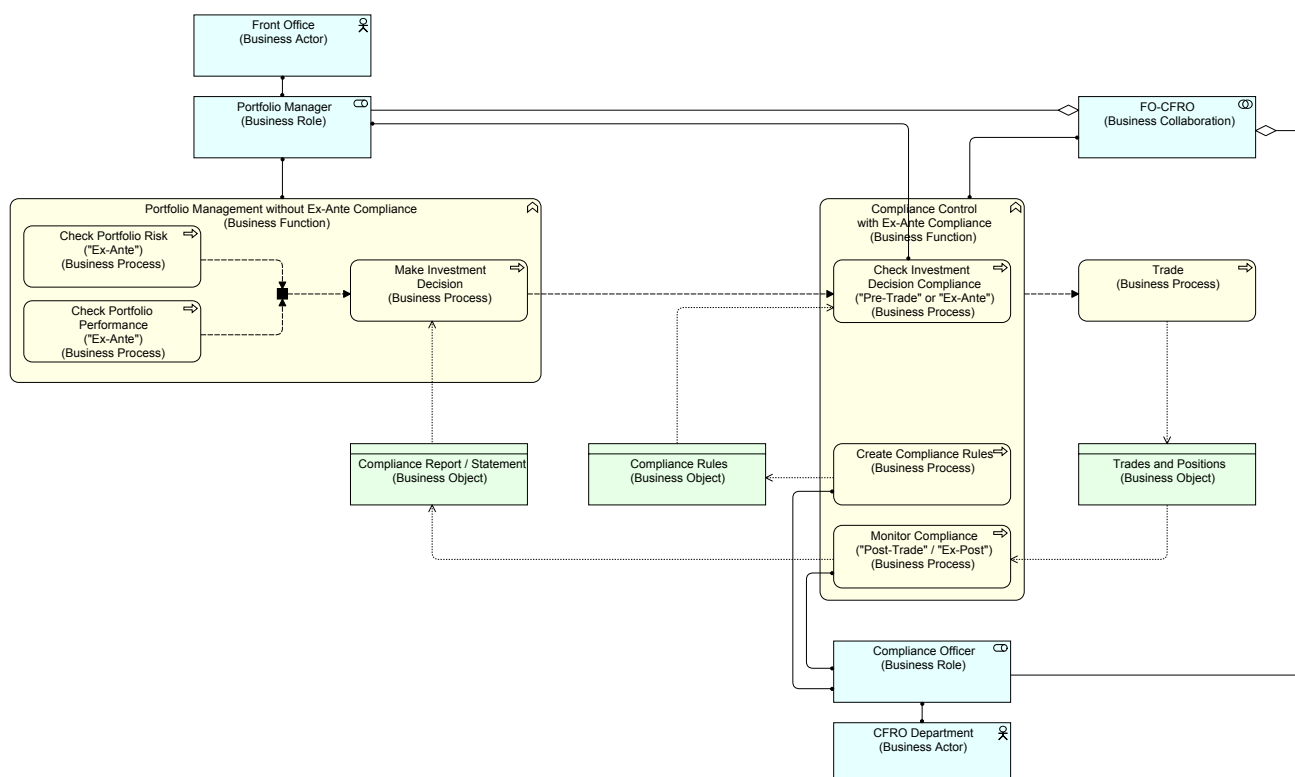
**View 134.** *Simplified Portfolio Management with Ex-Ante Compliance part of the Compliance Control function but performed by the Portfolio Manager role*



**View 135.** *Simplified Portfolio Management with Ex-Ante Compliance part of the Portfolio Management function but performed by the Compliance Officer role*

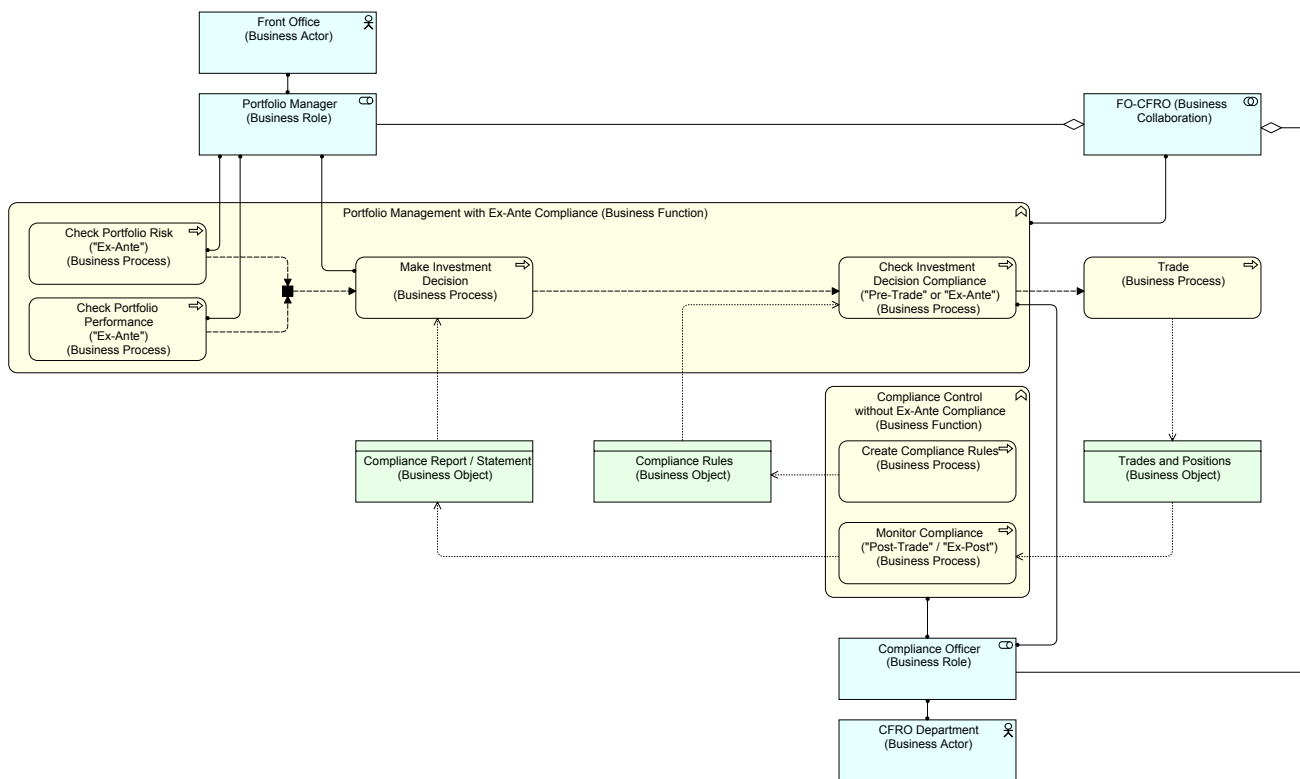
of the compliance function. Well, that is possible as well. It looks like View 134 on page 79. The 'compliance' function now has two processes that are performed by the compliance officer and one that is performed by the portfolio manager. View 135 contains the reverse split: the compliance officer role performs a step in the portfolio management function. ArchiMate has no formal problem with writing it this way (though it may be frowned upon), we look at Business Functions and Business Roles inde-

pendently and we can have a Business Function where the internal processes are performed by multiple roles. The question that arises of course is: what Business Role needs to be assigned to the overall 'compliance' Business Function in View 134 on page 79? We have a function, but who performs it? It must be a single role, according to ArchiMate. The solution is to use a Collaboration, as in View 136.



**View 136.** *Simplified Portfolio Management with the Compliance Control function performed by a Collaboration of Portfolio Manager and Compliance Officer*





**View 137.** Simplified Portfolio Management with the Portfolio Management function performed by a Collaboration of Portfolio Manager and Compliance Officer

Summarizing: if you want all ‘compliance’ activities in one Business Function, but not all processes of that function are performed by the same role, you end up with a Collaboration that is needed in your model to perform the function. To make the set of examples complete, have a look at View 137. Here the Portfolio Management function is performed by a Collaboration

**Question 6.** Can you spot the incorrect ArchiMate in View 136 and View 137?

**Answer 6.** Collaborations may only be Assigned-To Interactions

Though ArchiMate suggests that a function should be performed by a single role, this role can well be a Collaboration (which itself is a type of role), though the function then becomes an interaction. Function and *actual* Role being disjunct could therefore very well be the case if you have other characteristics that drive your grouping into Business Functions. I personally would advise against it, though. Because if your Business Function landscape differs widely from your Business Role landscape, it will be very difficult for everybody to keep track. In other words: being performed by a single recognizable role is an important potential dividing characteristic for dividing your landscape up in Business Function, just as ArchiMate suggests.

Now, if you step back from all the possibilities of modeling for a while, and you look at it from an Asset Management perspective, some of these models will be considered as plain silly by investment professionals. But that is in fact more about how they are used to look at things, than that the models are ‘wrong’.

Personally, I would indeed use the ‘single Business Role’ as an important characteristic for separation into Business Functions. So, at an Asset Manager where the portfolio

manager does ‘Ex Ante’ compliance checking, I would go for separating into Business Functions according to View 133 on page 79 (and not like View 132). But if the compliance officer does the ‘Ex Ante’ compliance checking, I would probably *still* go for View 137 and not View 134, the reason being that the whole chain of portfolio management to trading has quite a different beat to it (continuous, straight through processing) than the batch-like daily measurement of *the rest of* compliance, let alone the once in a while change (creation) of compliance rules. In other words: Business Role is an important characteristic for separation into Business Functions, but it is not by definition the *only* characteristic. Time patterns may be an important characteristic as well. And there are more. In fact, ‘Ex Ante’ compliance checking is so weakly connected to the skills of either portfolio manager or compliance officer, if you choose View 133 on page 79 or View 137, your main division into Business Functions stays the same, even if the company decides to change the assignment of ‘ex ante’ compliance checking from a portfolio manager to a compliance officer (or the other way around).

So, here you see it again: there is no simple answer on how you should model, just as having a grammar does not mean you know which sentences evoke the best effect in your reader.

## 11.2 Concurrent Functional Landscapes

So, among the most difficult Enterprise Architecture discussions are discussions on separating your business layer into Business Functions. The previous example already gave an example on how something can easily (and properly) be seen as a part of two different functions. But even after you have created your ideal functional decomposition, keeping it for all uses may not be the best thing to do.

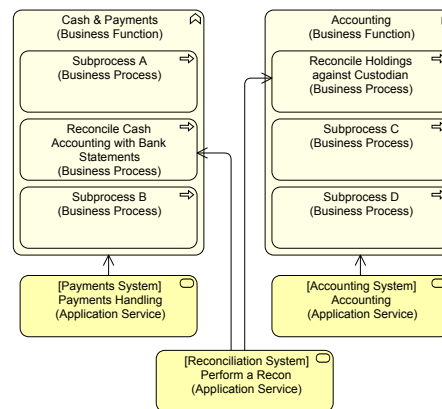
Take the following example: in Asset Management, we can have — amongst all our functions — the following two functional decompositions: ‘fund accounting’ on the one hand and ‘payments & cash’ on the other hand. Both generally have to do with business objects that are being held in the outside world: banks have the bank accounts where the cash is, custodians have the custodian accounts that hold the actual assets, e.g. stocks and bonds. If we buy stock from another party via the stock exchange, we enter into a deal, which is then confirmed by both sides and then follows ‘settlement’. Settlement means: we instruct our bank to send cash from our bank account to the other party’s bank account and they instruct their custodian to send stocks from their custodian account to ours. It often takes some time for this to be completed, say 3 business days. At the end of each day, our bank sends us a bank statement with all our transactions of that day and the end cash position on that account. The custodian does the same on our holdings. When we strike the deal, we put in our accounting system that we will receive the stocks in three days, and we also put there that we will have pay out cash which they will receive in 3 days. But in both cases, we also check afterwards if what has actually happened is the same as what we said would happen. Such checking is called ‘reconciliation’.

Functionally, we might have a ‘Cash & Payments’ Business Function and a ‘Fund Accounting’ Business Function. This is because handling custody and handling cash are quite different in processes, outside contacts, skills, etc.. So, our ‘Cash & Payments’ Business Function has a daily ‘Reconcile Cash Accounting with Bank Statements’ Business Process, while the ‘Fund Accounting’ Business Function has a ‘Reconcile Holdings against Custodian Records’ Business Process.

So far, so good, but reconciliation is a pretty basic set of activities: you have the records of someone else, you have your own records, you compare records and you handle the exceptions (called ‘breaks’ in recon lingo). In terms of IT support, the activities are almost the same, even if they differ substantially at the business level. Hence there are systems that in a generic fashion support ‘reconciliation’: comparing two sets of data and support the work flow of those checks and the handling of any finding.

So, when you let Enterprise Architects discuss the functional decomposition of your business, they — knowing how much each reconciliation technically and operationally looks like another — tend to want to group all reconciliation in a ‘Reconciliation’ Business Function. They look not just at the business layer, they are focused on the effects on the IT landscape and when you take those underlying operations into account, a single grouping of all reconciliation seems logical, especially if in the back of your head, you are already thinking forward to the process of selecting the right platform for ‘Reconciliation’. Feeling strongly, both sides, business and architecture, fight fanatically for what they see as the ‘correct’ separation in disjunct Business Functions.

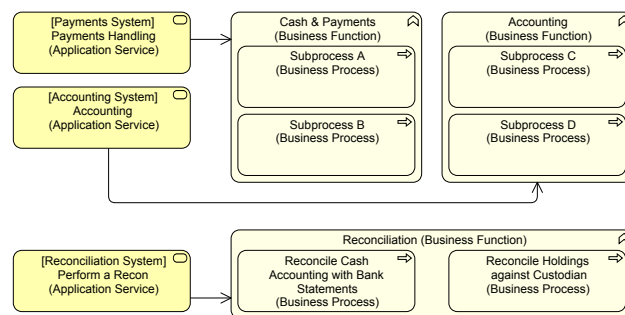
The simplified view without a separate Reconciliation Business Function looks like View 112:



**View 112.** *Cash & Payments function and Accounting Function both have a reconciliation process*

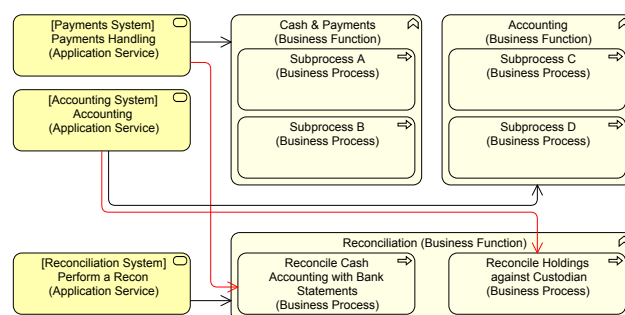
The Cash & Payments Business Function is supported by the Payments System, the Accounting Business Function is supported by the Accounting system. Both have reconciliation processes, which are supported by the Reconciliation system. Note: I am not saying Asset management is to be modeled like this, say, that the Cash & Payments Business Process does not need the Accounting system, I am just making a theoretical example as simple as possible. The real world is more complex.

When we model a separate Reconciliation Business Function, it might look like View 113:



**View 113.** *A separate Reconciliation Business Function*

I have, however, not been entirely complete. Because even if technically and operationally the actual data reconciliation only requires the Reconciliation system, handling ‘breaks’ (exceptions) will still require the Accounting System for holdings reconciliation and the Payments system for the cash reconciliation. If these are added, it looks like View 114:



**View 114.** *A separate Reconciliation Business Function still uses the accounting and payments systems for exception handling*

The added red Used-By relations show that the reconciliation (as seen from a *business* and not *IT* perspective) still

requires access to the accounting or payments system. In fact, in a derived way, these relations were already there in View 112, so one could argue from this example that you need not create a separate Reconciliation Business Function in your model, but that is not the point of this story.

The point is that if you look at it from a technical point of view, there is something like the bare technical and operational capability to handle reconciliations generically. In the views, it is what the 'Perform a Recon' Application Service supports. And if you start looking for ways how your IT is going to support your business, it is not a smart idea to handle both reconciliations separately. You might end up with an IT solution for the first one that is incompatible with the second one, forcing you to two systems or a migration. It is a smart idea to look at supporting the technical and operation aspects of doing a reconciliation in one go. The fact that business-wise the reconciliation process need access to their respective source systems to handle the exceptions does not mean that reconciliation itself is supported by either an Accounting or a Payments Application Service.

The two ways of looking at it are based on two different approaches to separation into Business Functions which we described above. One approach sees a Business Function akin to a capability to do something. The Business Processes make use of that capability or are made up of 'strings' of those capabilities. In that way of looking, the 'Reconcile Holdings against Custodians' makes use of both the Reconciliation capability and the Accounting capability. Both capabilities offer services to that Business Process. In this approach, the Business Processes are not the internal structure of a Business Function, the Processes are 'on top' of the Business Functions. Both ways of looking (as described here) are valid, in my opinion.

If you want to choose between making Reconciliation (or whatever your choice) a separate Business Function or not, ArchiMate helps in the sense that it says that a Business Function should be performed by a single Business Role. So, in case you have a clearly defined single Business Role that performs all 'reconciliations' (e.g. in an operational or technical sense that hands the results to other functions like Accounting or Payments to handle the exceptions), there is a decent reason to have a separate Business Function. But in a case like this, it is probably more important to combine monitoring and exception handling of 'holdings reconciliation' into one process that is performed by a single role: someone with knowledge of accounting and custodians and their processes. And the same is true for cash reconciliation. So, though we have reconciliation capabilities in our business, it will be performed in many places by many different roles in different contexts. Creating a single function of it at the business level which is performed by a single *real* role is not a good idea.

Now suppose we do indeed conclude that reconciliation is not a separate Business Function in our business because it is more like a capability that is part of different separate processes. How do we make sure we can take account of all those separate processes when we are looking at supporting the business with IT? Because in that case, it is useful to have a 'reconciliation function' that is to be supported by reconciliation Application Services.

The answer is that you nothing stops you to have a different separations into separate Business Functions for that specific goal. Let's go back to ArchiMate's description of what a Business Function is:

*A business function is defined as a behavior element that groups behavior based on a chosen set of criteria (typically required business resources and/or competences).*

And the specification continues with this explanation:

*Just like a business process, a business function also describes internal behavior performed by a business role. However, while a business process group's behavior is based on a sequence or "flow" of activities that is needed to realize a product or service, a business function typically groups behavior based on required business resources, skills, competences, knowledge, etc. that 'make up' the behavior.*

*There is a potential many-to-many relation between business processes and business functions. Complex processes in general involve activities that offer various functions. In this sense a business process forms a string of business functions. In general, a business function is behavior that delivers added value from a business point of view. Organizational units or applications may coincide with business functions due to their specific grouping of business activities.*

As described in Section 10.2 "Business Function or Business Process?" on page 72, if a Business Function is to deliver added value, it has to provide a Business Service. But as a Business Service is to be produced by a Business Process, a Business Function must have *internal* Business Processes that actually Realize those Business Services. Other, higher level, Business Processes may be based on that 'string of Business Functions', but that means the *internal* Business Processes of a Business Function are Used-By the (to the Business Function) *external* Business Processes.

But when we use a different set of criteria, we get a different grouping of behavior. If we want to group our business behavior on the basis of the criterium of IT-use, we get a different landscape. In that case, we might prefer the model where Reconciliation is a separate Business Function. In our example, if we follow the criterium 'competences' we get one landscape and if we follow 'IT-resources' we get another.

In summary:

- You divide business behavior up into Business Functions based on criteria;
- Different criteria produce a different landscape;
- Forcing everybody always in one specific landscape, produces irreducible conflicts. Hence, you might end up using multiple 'landscapes';
- Any landscape is based on a grouping of the internal Business Processes of the Business Functions, these must be the same for all landscapes to make sure all landscapes are not contradictory to each other.

I have named the separate IT-resource enterprise landscape the **BITMAP**: Business-IT Mapping.



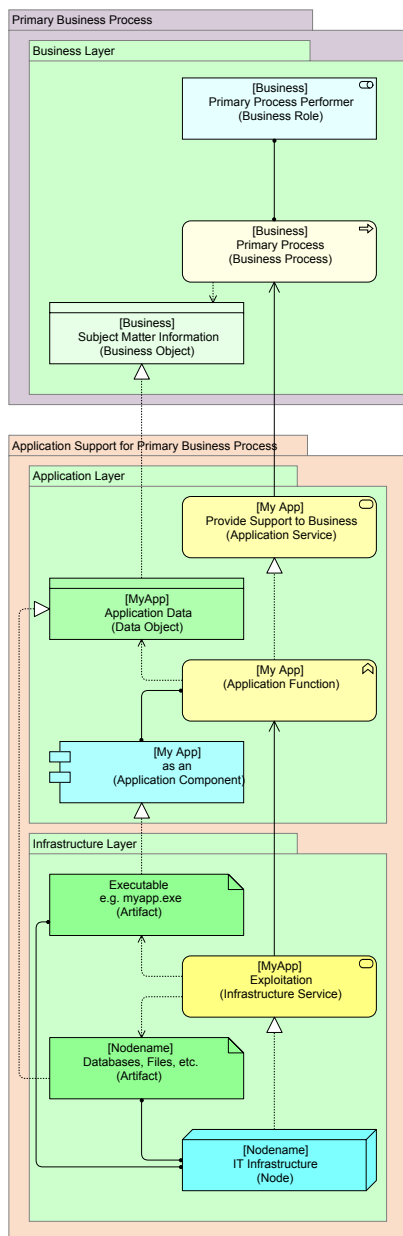
## 12. Secondary and Tertiary Architecture

### 12.1 Introduction

A while back, when we were building our Current State model somebody posed the question: How are we going to add the 'application owners' to our model? The question arose, because we had decided to use our Current State ArchiMate model as source for our CMDB on which the helpdesk software is based. In that database, the helpdesk people need to be able to find the owner of the application if something is wrong with it. In fact, our ArchiMate model became our CMDB.

Now the easiest way is to create a Business Role that you connect to the Application Component with an Association. That is the quickest way. But working with ArchiMate a lot influences your thinking. So, when somebody came up with the word 'owner', my thought was: "Wait a minute, that sounds like a *role*...". So, if owner is a Business Role, what Business Process is actually assigned to that Business Role?

What, in other words, is 'the owner's process'? What does an 'owner' *do* actually? And that made me think about other roles surrounding the use of IT: application manager, developer, exploitation manager, architect maybe even. The thinking resulted in an approach that I have christened 'Primary, Secondary and Tertiary Architecture'.



View 139. The Primary Architecture: Business uses an Application that uses Infrastructure

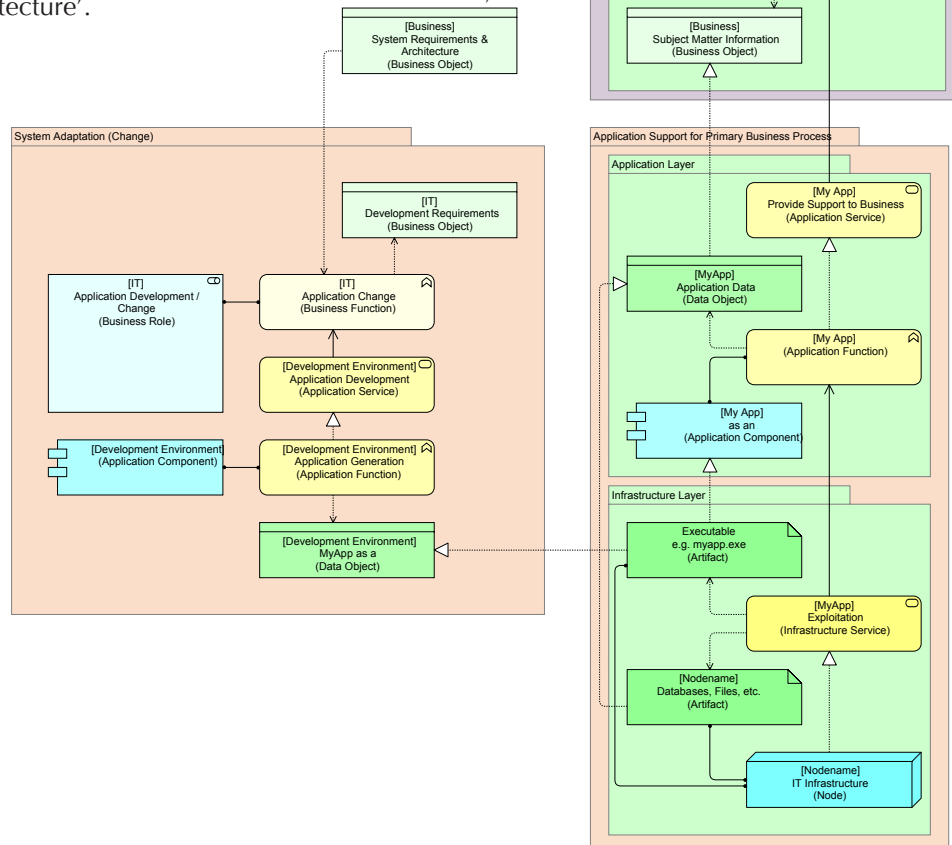
### 12.2 Primary Architecture

Primary Architecture is what it is all about and what is generally on everybody's mind and what we have been using as example most of the time. An example summary is in View 139.

This is what generally is the sole focus of Enterprise Architects: how the business processes are supported by IT. More or less, you see here a small summary of the essentials of the ArchiMate meta-model.

### 12.3 Secondary Architecture: Development

But as we all know, there is a lot that is needed for this 'Application Support for the Business' to work. Assuming we build our own application (comparable things happen when the application is not built but bought), we need to model the *development* of the application. Application development is a Business Process in itself, which as a result produces the application to be deployed and run. Suppose, this is a simple application and it is supposed to run in a Windows environment. What the development process *physically* has to produce is an executable ('.exe') file. That file is already in our Primary Architecture view. To *produce* that executable, the developers have their own Business Process,



View 138. The Primary Architecture with the Secondary Development Architecture

they employ their own Application Support, the latter generally some sort of Application Development Environment (ADE), an application that produces applications. This is shown in View 138.

I want to point your attention to the fact that the executable Artifact now realizes two objects at the Application Layer:

- For the primary business process, the Artifact Realizes an Application Component: an active object that supports the primary process with automation.
- For the (secondary) development process, however, that same Artifact Realizes a Data Object, that is created/updated by the development environment which supports the development process.

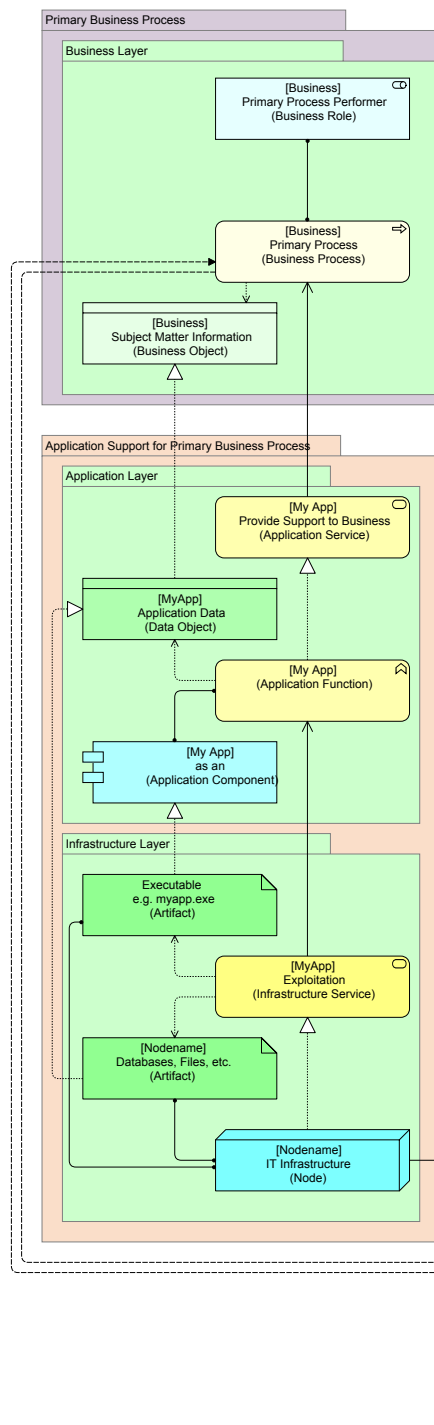
In other words, what for the business user is an application, is data for the developer. The user Uses the Application Component, the developer Accesses a Data Object. Looking at the development process, we see the development environment that is used to create the application. That application is used by the 'Application Change' Business Function which is Assigned-To the 'Application Development / Change' Business Role. Note: I have used a Business Function here, more or less because this behavior is not application-specific. I could have used a process as well.

Lastly, there are two Business Objects:

- The requirements for the application. This is an important input for the development process.
- The development requirements. There are some things the development function cannot handle. For instance, there may be policies regarding the development languages to use, because supporting every development language ever created is just too expensive and risky. Each of these languages comes with restrictions. If a business requirement would be in conflict with the restrictions of the development function, there is a conflict that needs to be resolved. We get back to these later.

## 12.4 Intermezzo

Firstly, I am fully aware that in most organizations it is already hardly common practice to model everything that is part of the *primary* architecture. So, modeling the development process is something that is generally thought of as 'nice to have'. I agree and that is why I call this 'sec-



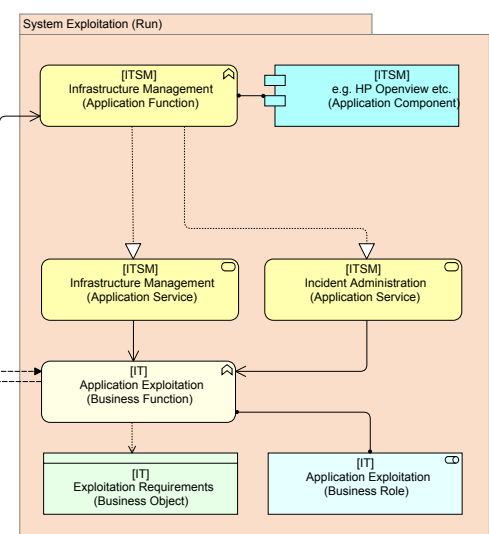
View 140. The Primary Architecture with the Secondary Exploitation Architecture

ondary'. However, the development role must see this as its own *primary* process and would do well to actually get it under control. Having said that, I am not expecting you to model every nook and cranny of your organization. I am using ArchiMate modeling here to make an analysis of the situation, in the end, I will present a short and usable summary that only slightly extends primary architecture.

Secondly: the labeling of the business layer objects has between the []-brackets the proposed part of the organization responsible, generally 'business' or 'IT'.

## 12.5 Secondary Architecture: System Exploitation

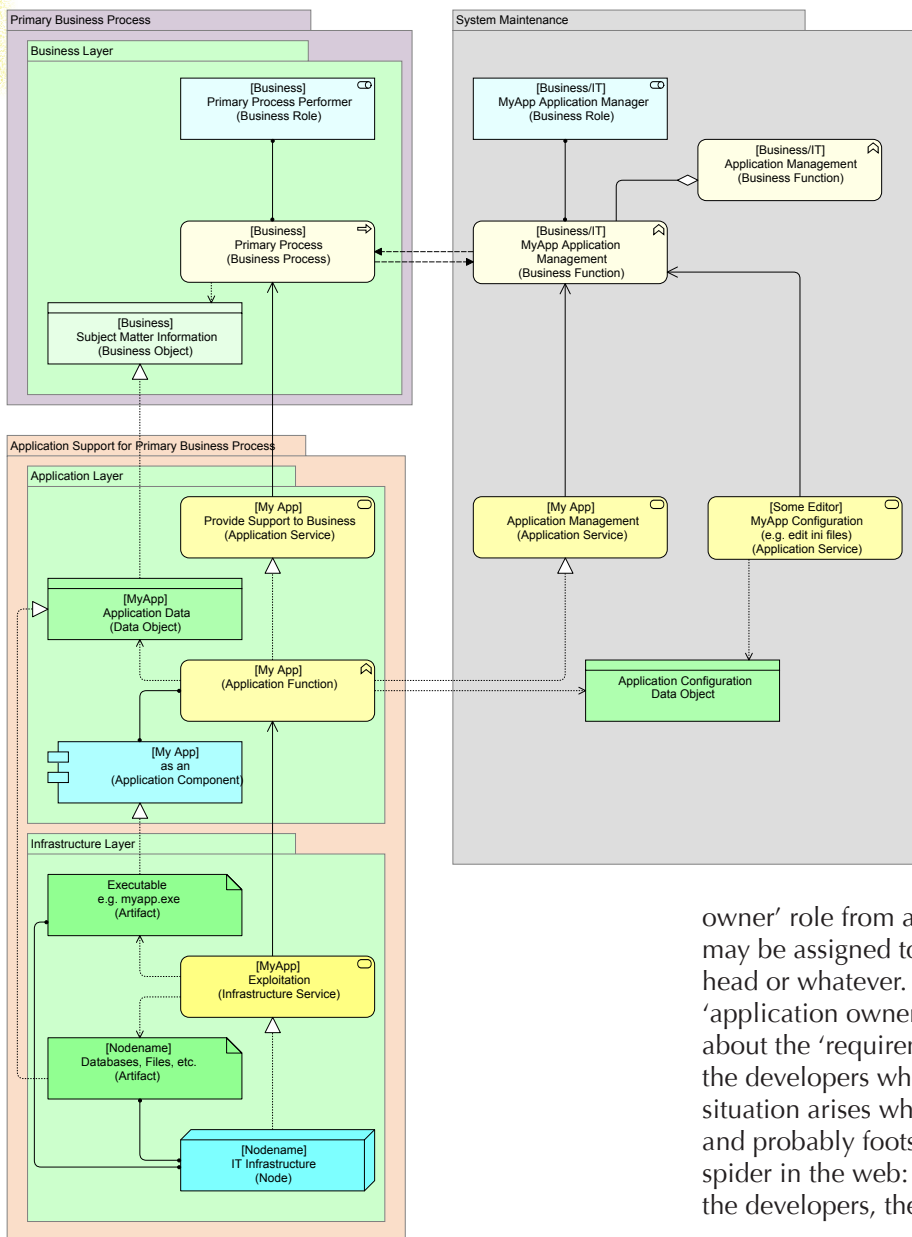
When the developers have delivered the application, it needs infrastructure to function. The application may be deployed on the hardware (the Node), but the 'run' people keep track of the infrastructure smoothly running, etc.. The 'run' people also have a help desk, where users can go to with problems. An example is modeled



in View 140. The 'run' people also have Business Processes that are supported by their own Application Services Realized by IT Service Management (ITSM) software. Two are modeled in the example:

- Infrastructure Management, which means keeping an eye on the infrastructure to check if it is still running properly;
- Incident Management to keep track of incidents, requests etc.

In the example, the Infrastructure Service that the Node Realizes for the Infrastructure Management software to use has been left out, what you see is the derived relation between Node and Infrastructure Management software. Additionally, the incident and request information and answers that flow from the primary process to the ex-



**View 141.** The Primary Architecture with the Secondary Application Management Architecture

exploitation process (and back) have been modeled as Flow relations.

In reality, all of this is far more detailed and complex, but the idea of this example is to show how the various aspects of using IT in your organization are related.

## 12.6 Secondary Architecture: Application Maintenance

The last aspect of Secondary Architecture is application maintenance. Application maintenance consists of things like managing settings of an application, such as user profiles, authorizations, or other maintenance jobs like management reports and such. Generally, in an organization, the larger applications have dedicated application managers who are responsible for these activities.

View 141 on page 86 contains an example. In that example, two patterns have been modeled:

- the application MyApp itself offers application management itself (e.g. screens to modify user's rights and so forth)

- some other (not shown) editor is used by the application manager to edit configuration files which are Accessed by the application.

Furthermore, the business users will communicate with the application managers, this has been modeled by a Flow relation between both Business Processes. And finally, the application management for MyApp has been modeled as being part of an overall Application Management Business Function.

## 12.7 Tertiary Architecture: Application Owner

It is time to add the role that started this analysis: application owner. The expanded example model can be seen in View 142. We return to the original question: "What does an application owner do?", or — since the 'application owner' is clearly a Business Role — "What is the application owner's process?". Here you need to separate the 'application

owner' role from all the other roles that actual person may be assigned to, e.g. information manager, department head or whatever. If you think about it that way, than the 'application owner' is above all about one thing: deciding about the 'requirements'. The 'application owner' may tell the developers what to build (remember: a comparable situation arises when systems are bought instead of built) and probably foots the bill. The 'application owner' is a spider in the web: he or she communicates with the users, the developers, the application managers, the infrastructure managers, all of which have a role to play with regards to the requirements. The users want a change? That is a change of the

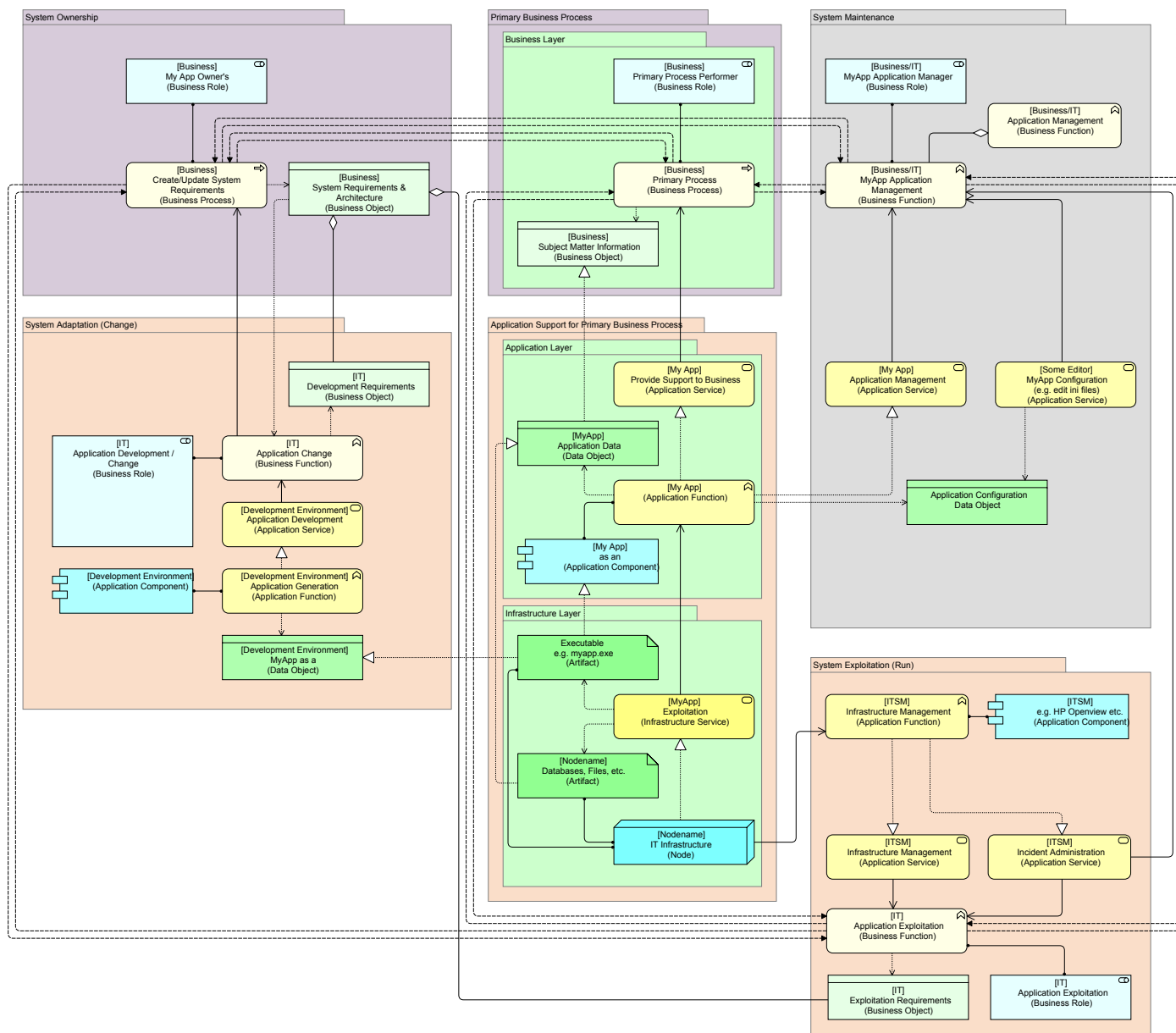
requirements that has to be decided upon by the 'application owner' role, all the while communicating with other stakeholders like developers and infrastructure managers. Infrastructure managers have their own requirements and so have the developers, their requirements have to be aggregated into the system requirements as well.

You can see there are more spiders in this web: the application managers communicate with all stakeholders and so do the infrastructure managers. After all: who are they going to call when there is a problem with the application which needs a change of that application?

## 12.8 Tertiary Architecture: other roles

We're almost done with this description. In View 143 on page 88 the final expansion of the example is shown. It adds a few other tertiary roles in summary form. These are: Process Owner, Data Owner and Enterprise Architecture. Remember, these are *roles* (the actual actors may overlap with other roles). They are shown here primarily because





**View 142.** The Primary and Secondary Architectures and the (Tertiary) Application Owner Architecture

they also have requirements that have to be taken into account by the application owner.

The 'process owner' is responsible for the process description and decides how a business process should run. Often, the 'process owner' is the same as the 'process executive' (which is not shown in this example), the one who is responsible for the execution of the process. Think of a department head who is responsible for the execution of the business process while a business control department may in the end be responsible for way the process must be executed (the process description). The user generally is a subsidiary of the process executive, and often either the process executive or some 'information manager' of his unit fulfills the role of application owner. Anyway, the process description is part of the requirements for an application, as the application is there to support that process after all.

The Data Owner may set policies for data, like retention, security, etc. This role also adds requirements.

Enterprise Architecture may set policies and goals for the way applications are set up and how they interrelate, an-

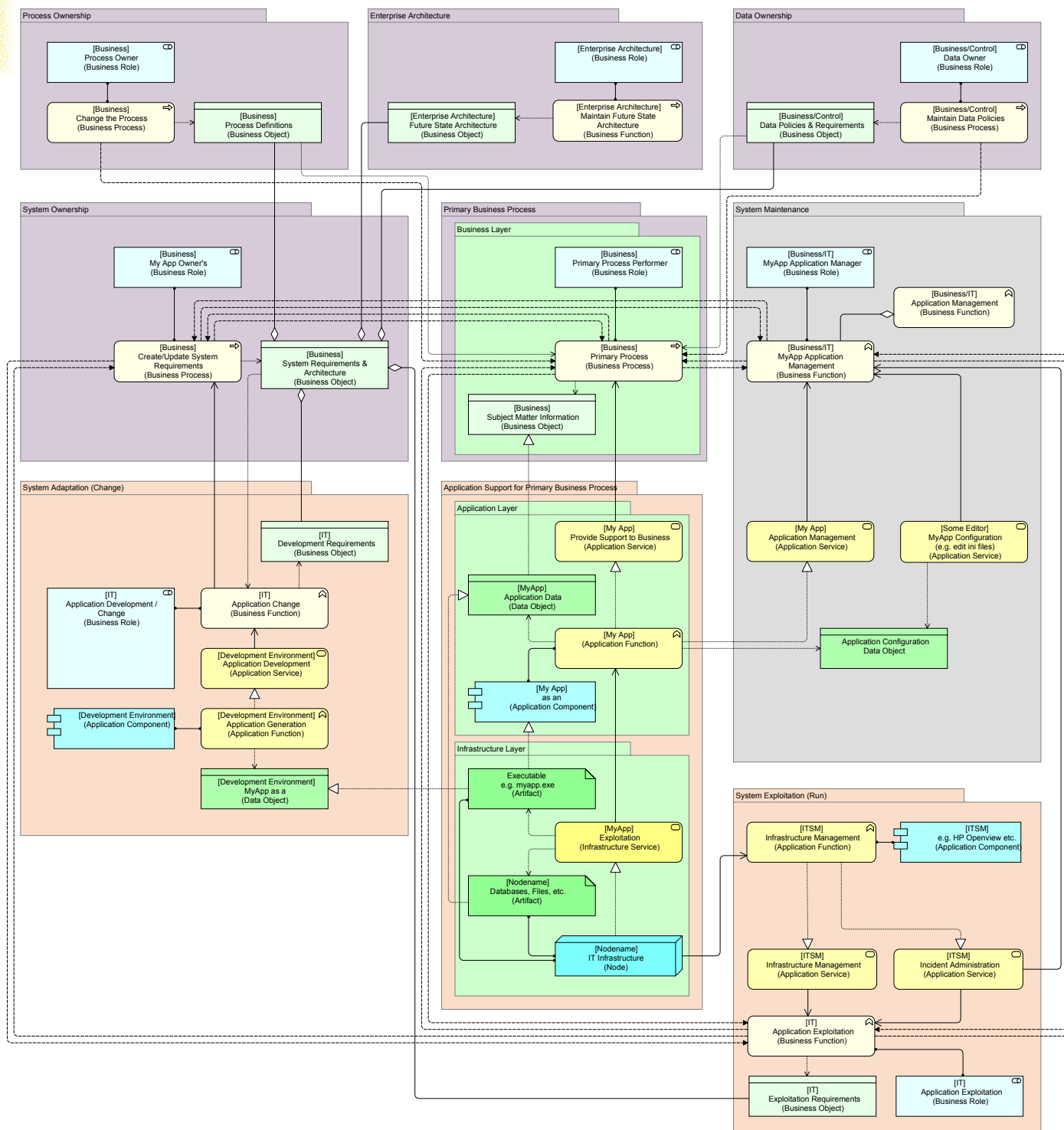
other addition to the requirements an 'application owner' has to manage.

Finally: the purple groupings are clearly 'business', the orange ones are clearly 'IT' and the grey one can be either.

## 12.9 Making it Practical: shortcuts

Now if you're in an average organization, you are probably far from realizing the complete model coverage of primary, let alone secondary or tertiary architecture. Maybe, when our discipline matures in a few decennia, it will be normal to do so. But for now, if you have to model all the secondary and tertiary aspects, it is probably a bridge too far. However, it might be good to model the most important roles regarding IT without fully modeling their processes and relate them to the primary architecture. A following set of 9 roles (as seen in View 144 on page 89) may be sufficient in most cases:

- **Run Manager.** This is the role that is responsible for the smooth maintenance and running of the infrastructure. This role may be accountable for any SLA that has been agreed on Infrastructure Services, hence we

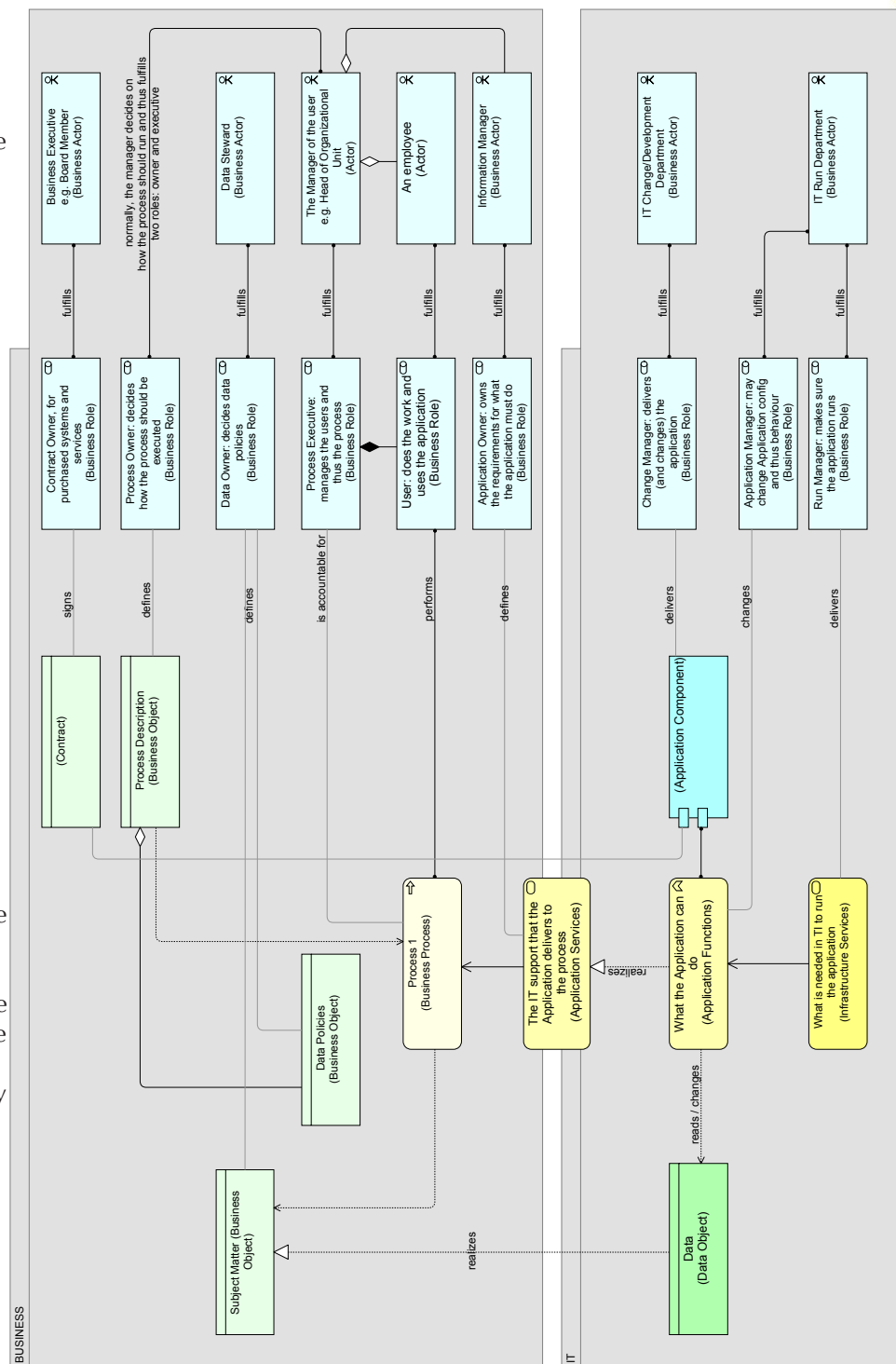


**View 143. The Full Picture: Primary, Secondary and Tertiary Architecture**

- model this as an Association Relation from the Run Manager Role to the Infrastructure Service. The role is generally fulfilled by the IT department.
- **Application Manager.** This is the classic application manager, the role that changes application settings (e.g. adding/deleting users or authorizations, changing application settings). It is the application functionality that is changed by this role, so we Associate it to the Application Function. This role is in practice fulfilled by many different actors in organizations. Some application managers are part of a business function, some are part of the change/development function, some are part of an IT delivery organization. In View 144, the role is Assigned to the IT Delivery organization.
- **Change Manager.** This role is responsible for the delivery of the application. If the application is built, these are the people responsible for development. If it is bought, these are the people doing the implementation and configuration. This role is generally fulfilled by IT-oriented (but business-aware) people.
- **Application Owner.** In the full Secondary-Tertiary approach, this role is responsible for the requirements of the system. What these requirements above all describe, is how the system is going to support the business. This is, so to say, the primary requirement, whereas the requirements from others (run, change, application management, architecture, etc.) are in the end always secondary. That primary part of the requirements is realized by the service the application

delivers (to the business), so we Associate the Application Owner role to the Application Service. This role is often fulfilled by a Business Executive or in larger organizations this is delegated to an 'Information Manager'.

- **User.** This is simple, this is the person using the system. We do not need an Association relation, because the User role is part of the primary architecture and the role is already Assigned-To the process that uses the Application Service.
- **Process Executive.** This is the role that is accountable for the execution of the process. It is generally fulfilled by the management of the user's department. The Process Executive is often not recognized in these descriptions and is often confused with the Application Owner. But in my setup, there is a difference. For instance, in a scenario with a major disturbance to the process because of IT failure, you inform the one responsible for the *process* (the User in this setup) and the one accountable for the *process* (the Process Executive, generally the management). The Information Manager as such does generally not have accountability for the process's execution, he has no operational role. Note: if these roles are fulfilled by the same person, you end up talking to the same person, but that does not mean that person is at that time fulfilling all his or her roles. In the example, I have made the Process Executive the manager of the Information Manager by Aggregating the latter under the former. What I have done here is using the Head of a Department and the Department interchangeably. How you model the manager role of a department is an interesting subject, by the way, which I'm not going further into here, I am just using this shortcut.
- **Data Owner.** Often, policies, regulations and laws govern data use. A separate owner of the data of a system may be used to design that role.
- **Process Owner.** This is the role accountable for the way a process is run. In the example, I have Assigned it to the Manager of the User, who also fulfills the Process Executive role. Indeed in practice, these roles are often fulfilled by the same executive. However,



**View 144.** Simplified way to model roles from the Primary/Secondary/Tertiary Architecture Analysis

in larger organizations, it may be delegated to, for instance, the Information Manager. Note that I have not been perfect with RACI (Responsible-Accountable-Consent-Informed) here, I am just creating a shortcut to model a few stakeholders into your Enterprise Architecture without modeling in full the Enterprise Architecture of those stakeholders.

- **Contract Owner.** Often, a system is (in part) purchased. There will be a contract that is important when looking at requirements etc. I have modeled this role separately and Assigned it to a top executive, who is generally supported by a Vendor Management function & department.



# 13. Modeling Risk & Security

## 13.1 Security Architecture

As far as Enterprise Architecture goes, one of the most underdeveloped areas is Security Architecture. It has been mentioned in most Enterprise Architecture frameworks and there are some new frameworks under development. Most of those that I have seen have a fundamental problem, which I'll illustrate with an analogy (always dangerous, analogies).

When Apple launched its own calendaring application (iCal) in 2002 they introduced a new architecture for the concept of a calendar. The calendaring application could hold multiple calendars. You could for instance load the calendar of all the sports matches you want to keep track of. Or you could load a view on another person's calendar (served by iCal Server) in your iCal window. Apple did (it now does, at least on Mac OS X) not offer the option to make appointments in your calendar *private*. Apple reasoned: it is not the *appointment* that is private, it is the *calendar* that is private or public. If you want to have appointments in your calendar that other people cannot see the content of, you need to make a second calendar for yourself with the private appointments.

Now, there is a basic flaw in this reasoning. As I am just a single person, I have only a single life. Which means that for people to be able to see if I'm available as a person, they have to see all my appointments. Hence, just giving them access to my public calendar and not to my private one does not work. Apple fixed this (in Mac OS X, not yet as I write this in iOS), probably under the market force of Microsoft Exchange, but it took them years to come around to the correct point of view (such inertia probably comes with the territory of being too often right ;-).

The analogy with Enterprise Architecture is that there is just a *single Enterprise reality* and thus that having a separate Security Architecture is nonsense. To be useful, Security Architecture must be fully integrated with your Enterprise Architecture, it must — ideally — be an *aspect* of it.

What Security Architecture in its core often is about are the following aspects of information:

- **Confidentiality:** Information is *only* accessible for those that should have access to it.
- **Integrity:** Information is correct. In practice this means that it cannot be modified undetectably.
- **Accessibility** or **Availability:** Information is available for those that need access to it.

These are the original CIA-aspects. Many frameworks add other aspects, like legally driven aspects such as:

- **Authenticity:** The source of information can be established with certainty.
- **Non-repudiation:** In practice this means that a party cannot deny certain information, e.g. having received something.

There are many more details with respect to modeling security, depending on the framework used. In fact, there

are many discussions on these (maybe even more than on Enterprise Architecture frameworks, who will say). All these aspects have an effect on *risks* that are related to handling information. E.g. a leak in your confidentiality might result in a financial risk, or a damage of your reputation or it might even land you in jail. Which brings us to the fact that these aspects used in Information Security are what in Risk Management are sometimes called *Control Objectives*.

## 13.2 Risk

If you move over to Risk Management, you find generally a division in three:

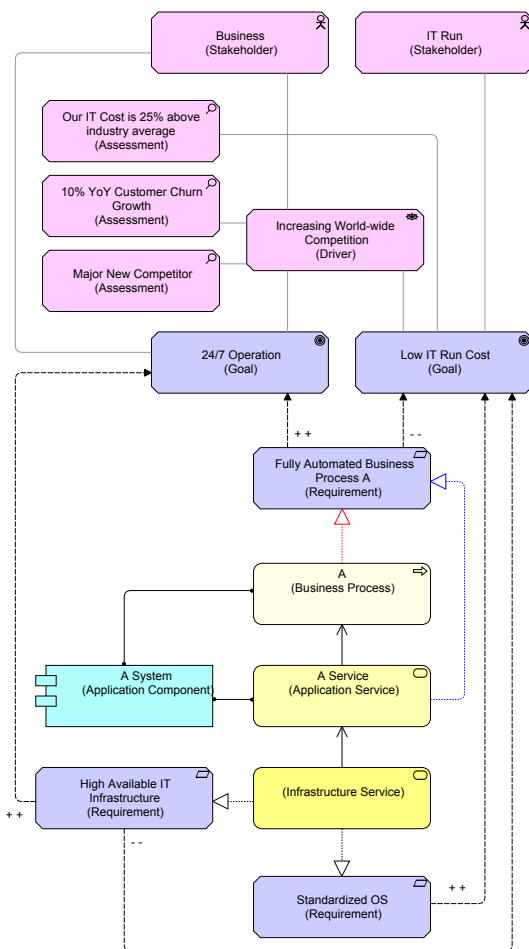
- **Risk:** The damage that can be done. The main aspects of risk are *impact* and *likelihood*.
- **Control Objective:** What state of affairs you want to achieve with regard to the Risks. In the most simple form, Control Objectives often take the form of the inverse of the Risk. But in more mature approaches, the Control Objectives are separate, as a single Control Objective may be a state of affairs that fights multiple Risks and for a single Risk, multiple Control Objectives may be important.
- **Control Measure** or sometimes **Countermeasure** or just **Control:** What you actually *do* to get to this state of affairs that is the Control Objective.

This division can be found in many frameworks, like ISO27001, ISA 3402. The main issue for us here and now is that if we are able to model the most important aspects of Risk Management in our Enterprise Architecture, we also have the framework to model the most important aspects Security Architecture. I am not saying that everything related to Security or Risk can be brought back to a few simple objects and relations. Risk and Security merit a far more comprehensive approach than that. But modeling Risk, Control Objective and Control Measures in your Enterprise Architecture may offer a good link between your Enterprise Architecture and the Security and Risk Management in your organization.

So, we come to the question if ArchiMate can help us adding Security and Risk Management aspects to our Enterprise Architecture. And — since ArchiMate 2.0 — it can. I am presenting to you here an approach (first collaboratively pioneered at my company by colleagues Jos Knoops and Roy Krout). But first we introduce ArchiMate 2.0's Motivation Extension.

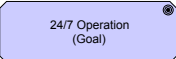
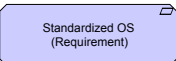

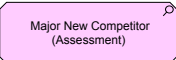
### 13.3 ArchiMate 2.0 Motivation Extension

ArchiMate 2.0 comes with an extension called the Motivation Extension. It contains several new concepts. It is easiest to illustrate (most of) them in an example as can be seen in View 115.


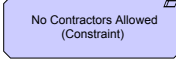
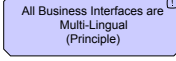


**View 115.** ArchiMate 2.0 Motivation Extension example

The new objects and relations are:

- 
**24/7 Operation (Goal)** This is a **Goal**, which is an end state that a stakeholder wants to achieve. In the example above, the Business wants to achieve 24/7 operations and the IT Run wants to lower IT cost.
- 
**Standardized OS (Requirement)** This is a **Requirement**, which is a obligatory aspect of what a system or process Realizes. In the example, there is a Requirement to use standard operating systems only in your infrastructure, to have high-available infrastructure and to have a certain business process fully automated.
- 
**Increasing World-wide Competition (Driver)** This is a **Driver**, which is something that drives change in the organization. These may be external (e.g. market forces, regulation) but also internal (e.g. the mission or vision of an organization). In the example, there is one driver: 'Increased World-Wide Competition'.
- 
**Major New Competitor (Assessment)** This is an **Assessment**, which is the outcome of an analysis of a Driver. E.g. if a driver is "customer satisfaction", a poll might result in an assessment. In the example, there are

two Assessments: a new competitor has arrived on the scene and we win/lose 10% of our customers each year (because they come to/from the competition)

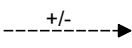
- 
**Business (Stakeholder)** This is a **Stakeholder**, which is a Role that is interested in achieving a Goal, or might be associated with a Driver or an Assessment. In our example, there are two Stakeholder roles: Business and IT Run (the responsibility for keeping the systems operational)
- 
**No Contractors Allowed (Constraint)** This is a **Constraint** (a Specialization of Requirement), which is a forbidden aspect of what a system or process Realizes. There is no Constraint modeled in the example, but it behaves like a Requirement.
- 
**All Business Interfaces are Multi-Lingual (Principle)** This is a **Principle**, which is a sort-of Goal that is a generalized Requirement. There is no Principle modeled in the example, but it behaves like a Requirement.

The concepts are not exactly, discretely separated. It is sometimes a matter of taste if something is seen as a Goal or a Driver. Having a certain strategic Goal as an organization may count as a Driver, it seems to me. And the divide between a requirement and a principle is not very clear too. This is not really a problem (we encountered some overlaps earlier, think Business Function versus Business Process), it then comes down to creating your own good patterns to use them.

The relations between the objects are generally Associations. The exceptions are that Realization is used to:

- let a Requirement, Constraint or Principle Realize a Goal
- let a Requirement or Constraint Realize a Principle
- let any core ArchiMate concept (except Value and Meaning) Realize a Requirement. In the example, an Infrastructure Service, an Application Service and a Business Process all Realize a Requirement. An interesting question is if you have — as is the case in the example — a Requirement like 'Fully automated Business Process A'. Is it the Business Process that Realizes this Requirement (in red) or the Application Service (in blue) or both? There is quite a bit of freedom in choosing your patterns.

and a new relation:

- 
**+/-** This is the Influence relationship. It is used to model the way Driver, Assessment, Goal, Principle, Requirement and Constraint can influence each other. Normally, a label on the relationship is used to denote the type of influence. In the example, the Requirement to use standardized operating systems in the infrastructure influences positively the 'Low IT Cost' Goal, but the 'High Available IT Infrastructure' influences that same Goal in a negative way. The 'High Available IT Infrastructure' Requirement positively influences the '24/7 Operations' Goal and the 'Fully Automated Business Process A' influences '24/7 Operations' positively but 'Low It Cost' negatively.

The Motivation Extension of ArchiMate 2.0 shows its origin: the developers wanted to catch the Architecture itself in their modeling language. Hence, the use of the concept

of Principle (which is a popular Enterprise Architecture concept) next to Requirement and Goal. What this new part of ArchiMate also shows is how different the world of human intentions and motivations is from the logical world of IT and how difficult it is to map one onto the other (“I have one negative influence and one positive. Now what?”). But what the Motivation Extension does offer us is a basic mechanism to model Risks, Control Objectives (including those from Security) and Control Measures and in that way link ArchiMate models to the world of Risk and Security:

### 13.4 Modeling Risks in ArchiMate 2.0

When modeling Risk, you can use the following mapping:

- A **Risk** is modeled using ArchiMate’s **Driver** object
- A **Control Objective** is modeled using ArchiMate’s **Goal** object
- A **Control Measure** (sometimes referred to as Counter-measure or just Control) object is modeled using ArchiMate’s **Requirement** object

We Associate a Process for which there is a Control Objective with that Control Objective. For instance, a payments process may have as Control Objective that it is not possible for a single person to authorize a payment. Such a Control Objective is then modeled as a Goal object that is Associated with the process it is a Control Objective for.

In the overview of View 145 on page 93 the patterns for modeling Risk (including Security Risks), Control Objectives (including Security Aspects of Information) and Control Measures are shown.

In the green grouping, we find a (primary) Business Process 1 that consists of three subprocesses Sub 1.a, Sub 1.b and Sub 1.c, where information flows from Sub 1.a to Sub 1.b and Sub 1.b to Sub 1.c.

Furthermore, 4 Risks have been modeled: Risk X, Y and Z and a Risk called ‘Security’. Note that these can in the real world be replaced by a complex tree of Risks, using for instance Aggregation and Composition relations. The Risks X, Y and Z are Associated with Control Objectives 1, 2 and 3 in an n:m way. For instance, Risk Y is Associated with Control Objectives 1 and 2, while Control Objective 2 is Associated with Risk X, Y and Z. The Risk ‘Security’ is Associated with three Control Objectives: Confidentiality, Integrity and Availability.

The Control Objectives are not only related to the Risks (red Associations) but also to the objects they are Control Objectives of (blue Associations). In the case of Control Objectives 1, 2 and 3 and Control Objectives Availability, the Control Objectives are Associated with Business Process 1. In case of the Control Objectives Confidentiality and Integrity, they are Associated (blue Associations) with the ‘Subject Matter’ Business Object they are Control Objectives of. Note that the color of the Associations here is only to make the explanation easier, I am not advocating per sé that blue Associations are used for linking Risks to Control Objectives and blue Associations for linking Control Objectives to the objects they are Control Objectives of. But to be honest, I find the idea attractive, as you can easily focus on the specific type of relation in a view.

Control Measures (or just ‘Controls’) contribute to the Control Objective. This has been modeled by using the Influence relation from Control Measure to Control Objective. For instance, the ‘99% uptime’ requirement (Control Measure) for the Infrastructure Service contributes positively to the ‘Availability’ Control Objective.

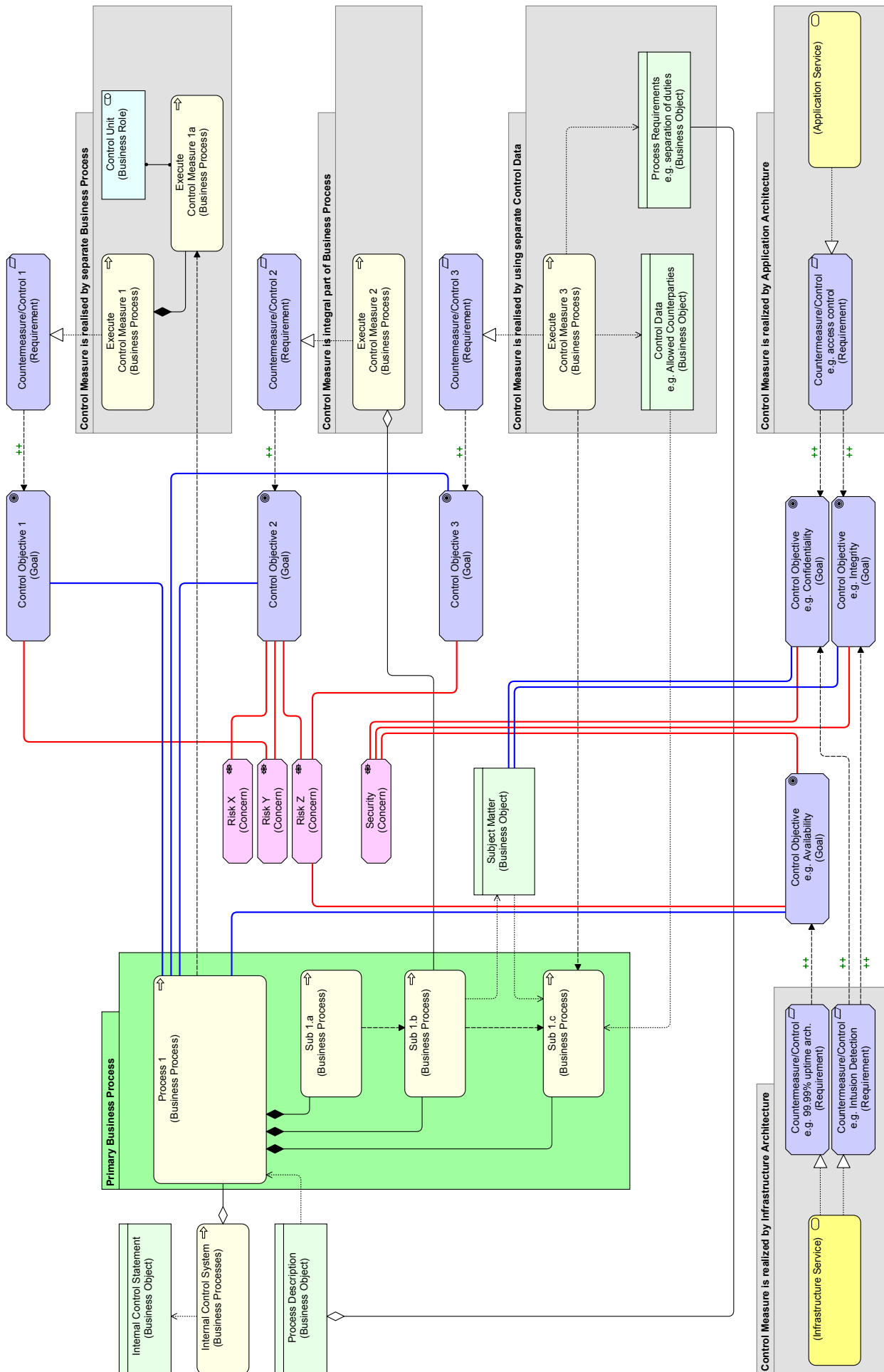
All Control Measures are either Realized by a Business Process, by an Application Service or by an Infrastructure Service. For the last two holds that the Control Measures are requirements to the service.

Three patterns are shown how Control Measures can be Realized by Business Processes:

- First, there can be a separate process that Realizes the Control Measure. This can for instance be a separate audit process. This is the case for Control Objective 1 in View 145. The audit process requires a flow of information from the audited process and this is true for any external process that realizes a Control Measure for a core process.
- Secondly, the actual Business Process that the Control Objective is about might have to be *changed* because of the Control Objective. For instance, there might be an extra step in the Business Process to perform a check or an authorization. In View 145: This extra step Sub 1.b is really part of core process 1. To denote that that part/subprocess is there *because* of the Control Measure, it has also been made an Aggregate child of the (possibly unrelated) collection of process steps (subprocesses) that together Realize the Control Measure. This aggregation therefore is a collection of all the changed parts of processes that have been changed because they are part of what Realizes the Control Measure.
- Thirdly, a separate process may be responsible for information that the core Business Process. This can be in two different ways:
  - \* First, a process may produce information that is used in another process. For instance, as a Control Measure to prevent trading with non-allowed parties or in non-allowed countries, the control function may produce a list of allowed parties or countries that is then used in the trading process.
  - \* Secondly, a process may provide requirements for how a core business process is run. For instance, requiring a four-eye principle on certain activities. Such requirements become part of the process’s documentation and adhering to the requirements can become again part of an audit type realization.

The first pattern is the pattern that also offers the proof that Control Measures have been taken and that therefore Control Objectives will be met.





**View 145.** Five ways (patterns) the Enterprise Architecture can Realize Control Measures/Controls that contribute to Control Objectives that are associated with Risks

# 14. More Platform Thinking

## 14.1 Low Level Data Entry

Recapitulating our approach with respect to software:

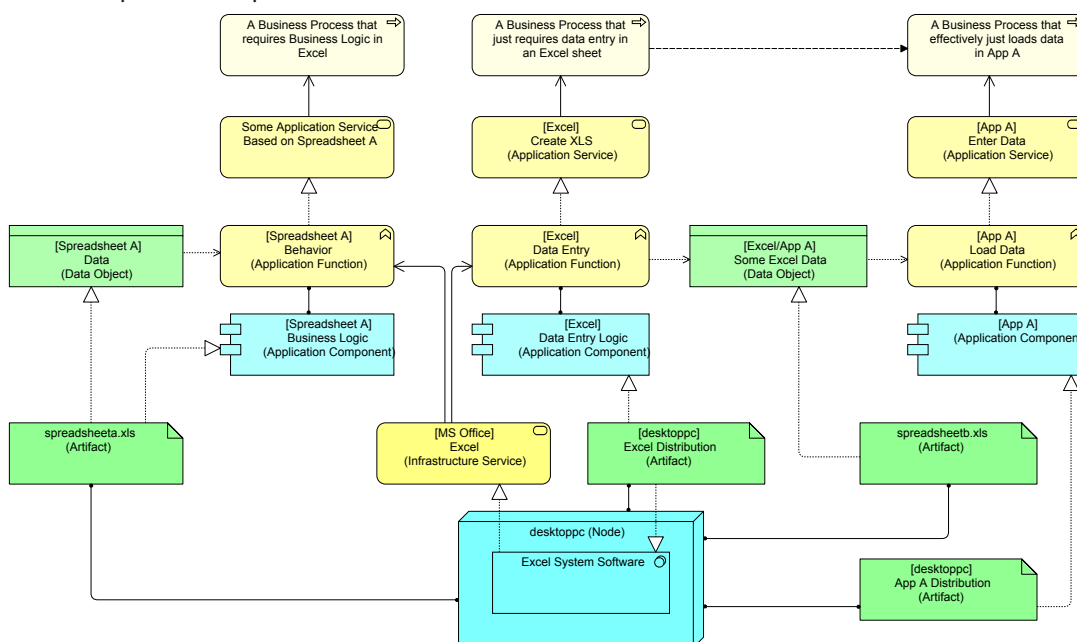
- If it has business logic, it is an Application Component. As a result, that spreadsheet your business uses is an Application Component and Excel itself is System Software that Realizes an Infrastructure Service (see Section 7.3 on page 46);
- If it does not have business logic, it is either something very generic, e.g. a database system or it is a platform of sorts (see Section 7.5 on page 49) or both.

But consider the following scenario: you have an application (App A) that requires you to load certain data via an Excel spreadsheet document. That is, you need plain vanilla data entry in Excel to create that .xls file. That file is then later loaded into the application. You do not use Excel as an Infrastructure Service *platform* to run business logic in an Excel spreadsheet, you create the spreadsheet using Excel as the *application*.

Since that System Software cannot Realize an Application Service itself in ArchiMate, it cannot be used directly by the Business Process. I suggest you do not read on immediately, but instead think yourself for a while how to solve the puzzle.

Now, that you've thought of a solution, I'd like you to go back to Section 9 "An Example from the Real World" on page 65. What happened there was that a platform was *both* used as Infrastructure Service for some sort of logic written for that platform (e.g. a Scheduler or Transporter workflow) *as well* as its own maintenance tooling. So:

- The system's distribution Artifact Realized the System Software that Realized the Infrastructure Service (the *platform*)
- The system's distribution (Artifact) Realized the Application Component that was the *maintenance/development* part of that platform.



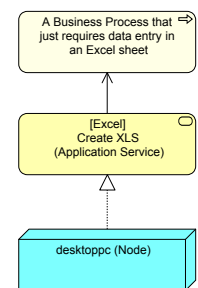
View 146. Using Excel for Data Entry for a Business Process

Actually, this is a pretty common pattern if you reason from the distribution Artifact. Now, go back to the Excel distribution (in fact MS Office, but that is not an important distinction). That distribution Realizes the Infrastructure Service for the spreadsheets that run in the platform. But that distribution *also* Realizes the Excel 'maintenance/development' Application Component just like in the example of Section 9. Summarizing: Excel is not just an Infrastructure Service, but it is *also* a *development environment* (in case you write business logic in Excel) and a *data entry environment* (in case you have to enter just data).

So, in those cases that data entry in an Excel spreadsheet (say a spreadsheet with user names and authorizations that some SaaS service requires to be uploaded to be able to set the users and authorizations) is an essential step in your business process, something you should not leave out because it is very important, you can use for instance a basic '[Excel] Spreadsheet Maintenance (Application Function)' in your landscape. Worked out in detail the reasoning can be found in View 146. (Incidentally, I would personally leave the Application Component, the distribution Artifact and maybe even the Application Function for Excel's 'maintenance' mode out of my model. Just use the '[Excel] Create XLS (Application Service)' and leave the rest out. In fact the solution to model Excel to create a file becomes pretty simple. It is shown in View 147. The argument for the Realization relation from Node to Application Service is left as an exercise for the reader (I always wanted to write that since having to study Gasciorowicz...)

What this also shows is that Excel is its own development environment. Creating an Excel spreadsheet is data entry when it is just entering data in cells for other applications to read. Or it is just a different way of writing a different

kind of document, e.g. a report. But as soon as you start using macros and formulas, you are in fact using the Excel development environment to write an *application*, which is an Excel spreadsheet that runs using the Excel Infrastructure Service as in 7.3 "Modeling Spreadsheet Applications" on page 46. This is not different from the maintenance environments shown in Section 9 on page 65.



View 147. The 'Create Spreadsheet' Excel Application Function

This page intentionally not left blank



# Model Use & Maintenance

U

U

e

U

U

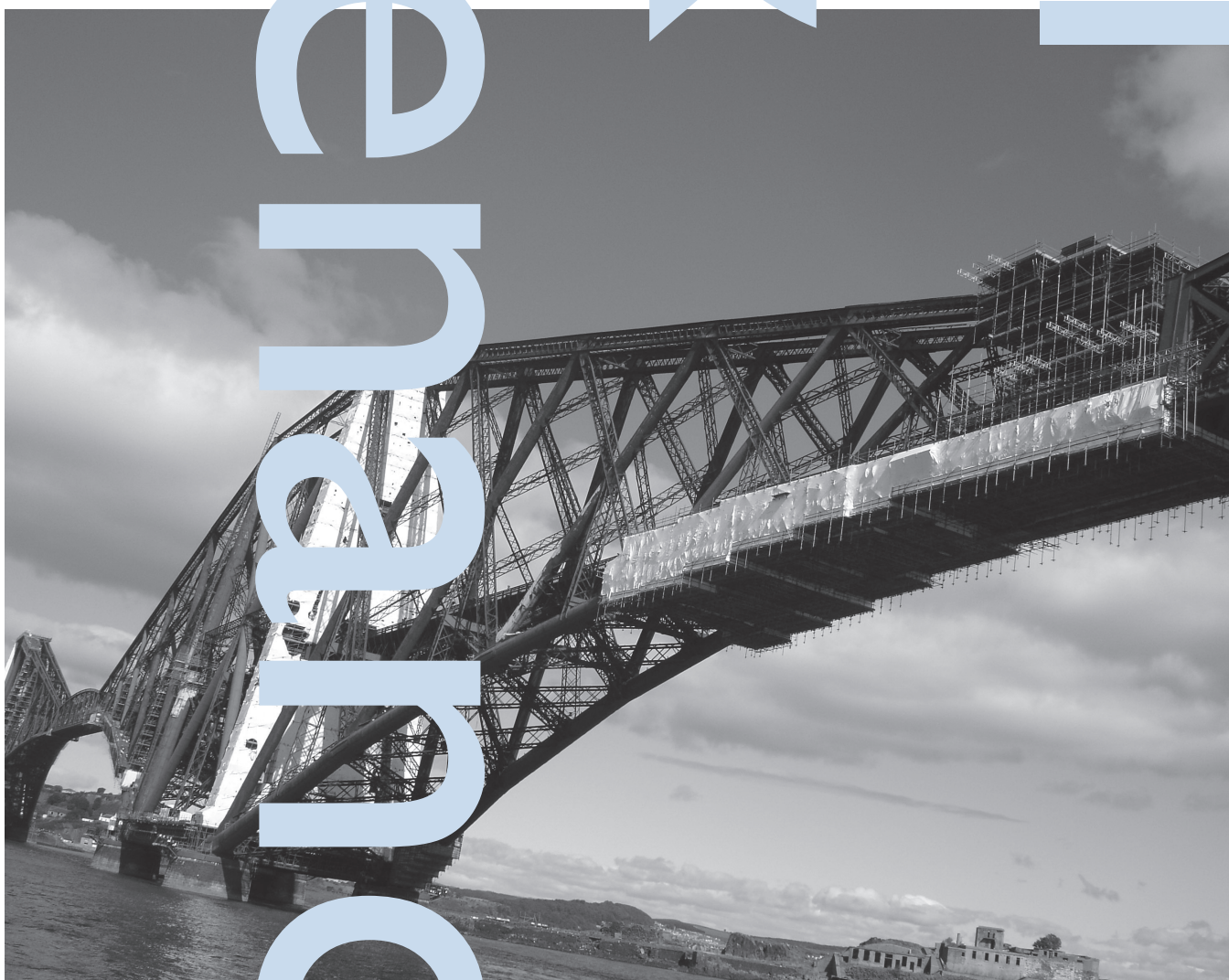
U

U

∞

U

U



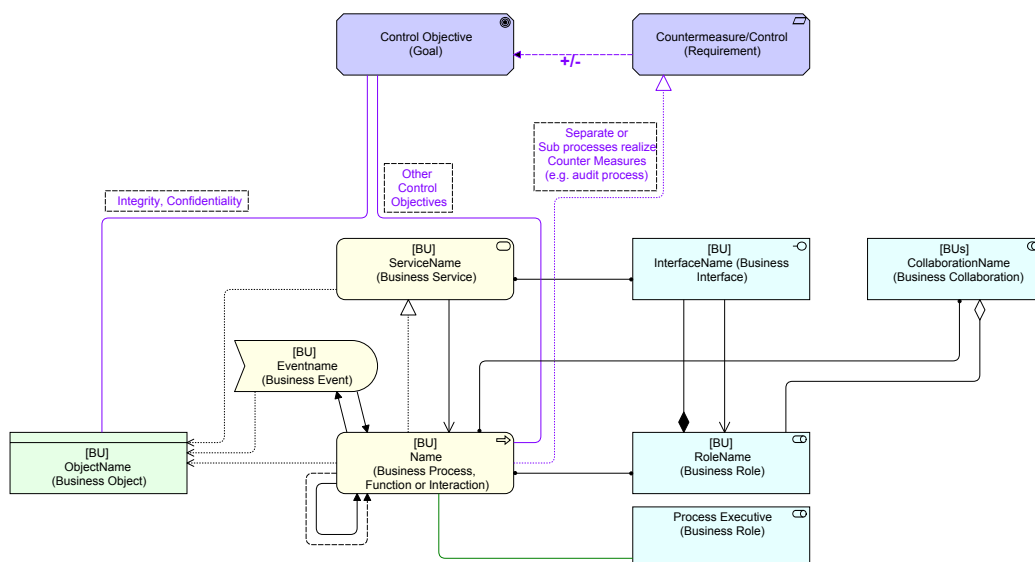
# Model Use & Maintenance

## 15. Construction & Use Views

When you get *really* serious with ArchiMate, and you model your entire Current State in ArchiMate, your models will get pretty large. My own estimate of our model when we are complete, with risks, secondary and tertiary architectures and more lies around 20,000 objects and 25,000

We decided to use a couple of ‘view types’ to maintain our current state model (and models for large projects). I think our end result is not yet finished, as we have found that maintaining construction views is not enough to publish the model, so we need other views, specialized

for a specific audience. Using the viewpoints mechanism from the ArchiMate standard also doesn’t cut it for us. So, we created a couple of ‘view templates’ to divide our landscape in and that we use to model our large environments, especially the current state. Note, these views are not truly ArchiMate ‘view-points’. ArchiMate viewpoints restrict a view to certain object and relation types. Our views are not quite like that. For instance, the Application Architecture View does not allow all Business Roles, only explicitly the Change Manager, Application Manager and Application Owner roles. You can see a version of the

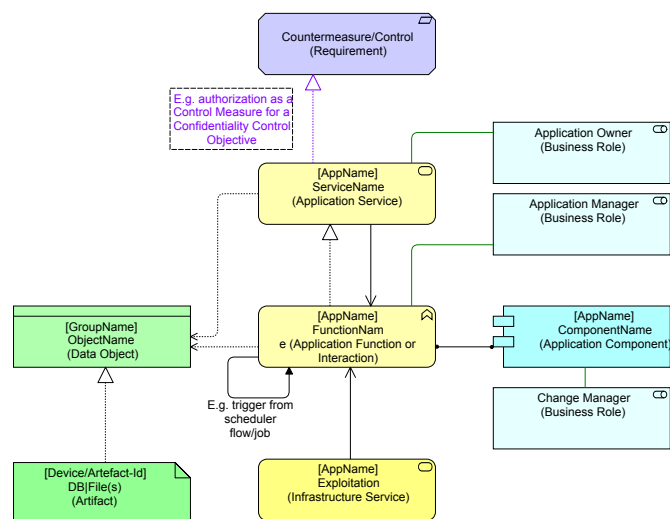


View 148. Large Model Construction - Business Architecture View

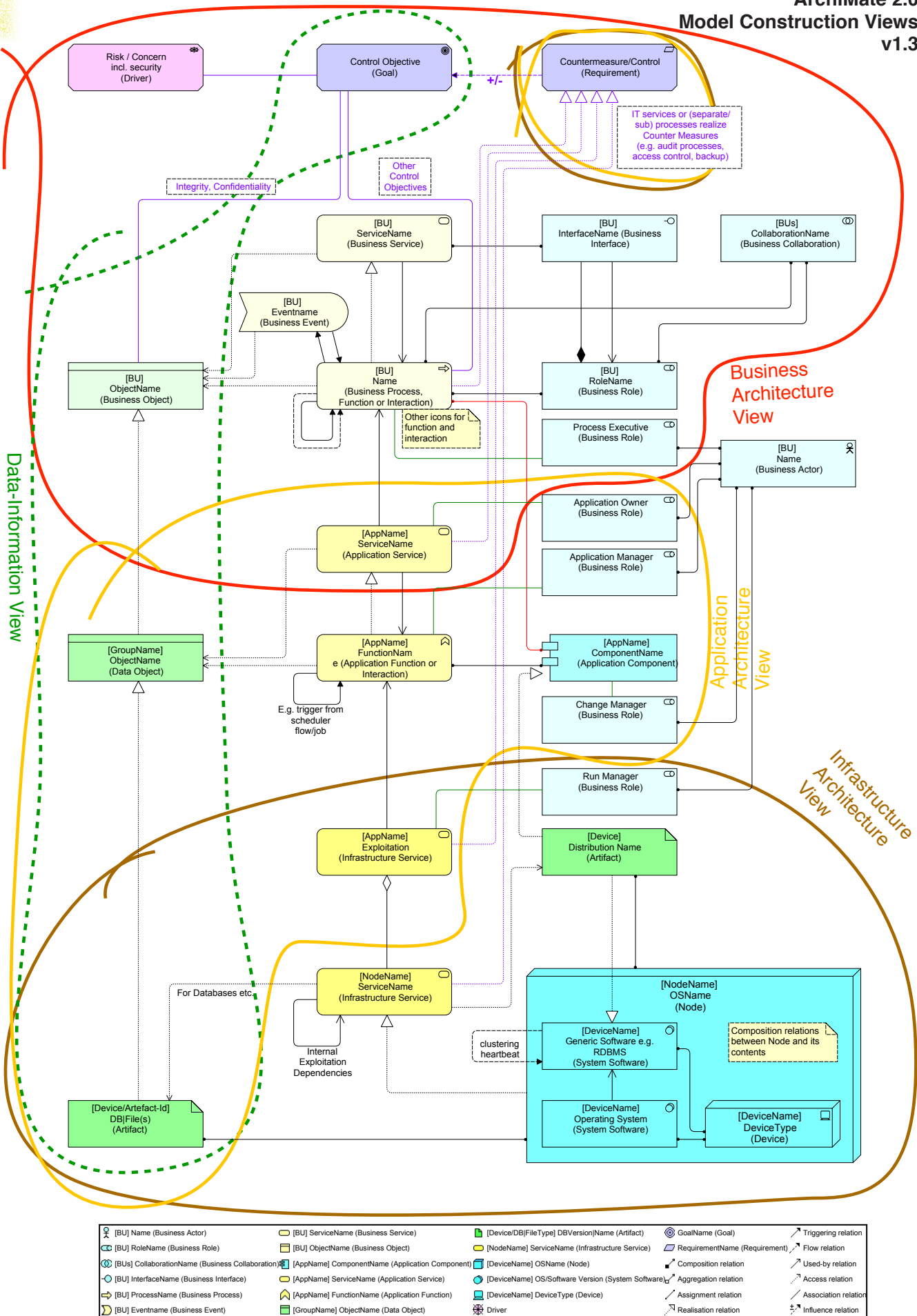
relations. This sounds unmanageable, but it isn’t if you do it right (this comes from experience, we build & manage this with roughly one FTE after doing an initial stint with two FTE). And besides, if you have such a model, it will save a lot of work elsewhere in your organization. Now, apart from the right patterns to which you should stick religiously, you will need a way to model this. One big view of 20,000 objects and 25,000 relations doesn’t cut it.

Before I describe the details, there is one important point and you might to choose differently. We decided *not* to model Interfaces. As each Interface is a one-on-one for the Service it is Assigned-To, we decided we could keep our models simpler (they are complex enough as is) by leaving the Interfaces out. My philosophy is that behavior is the core of what happens in the organization and we model around that. This is also the reason we chose the original color setup (with a twist). Basically, it was all about keeping things manageable in the light of the huge scale we were planning to do our modeling in.

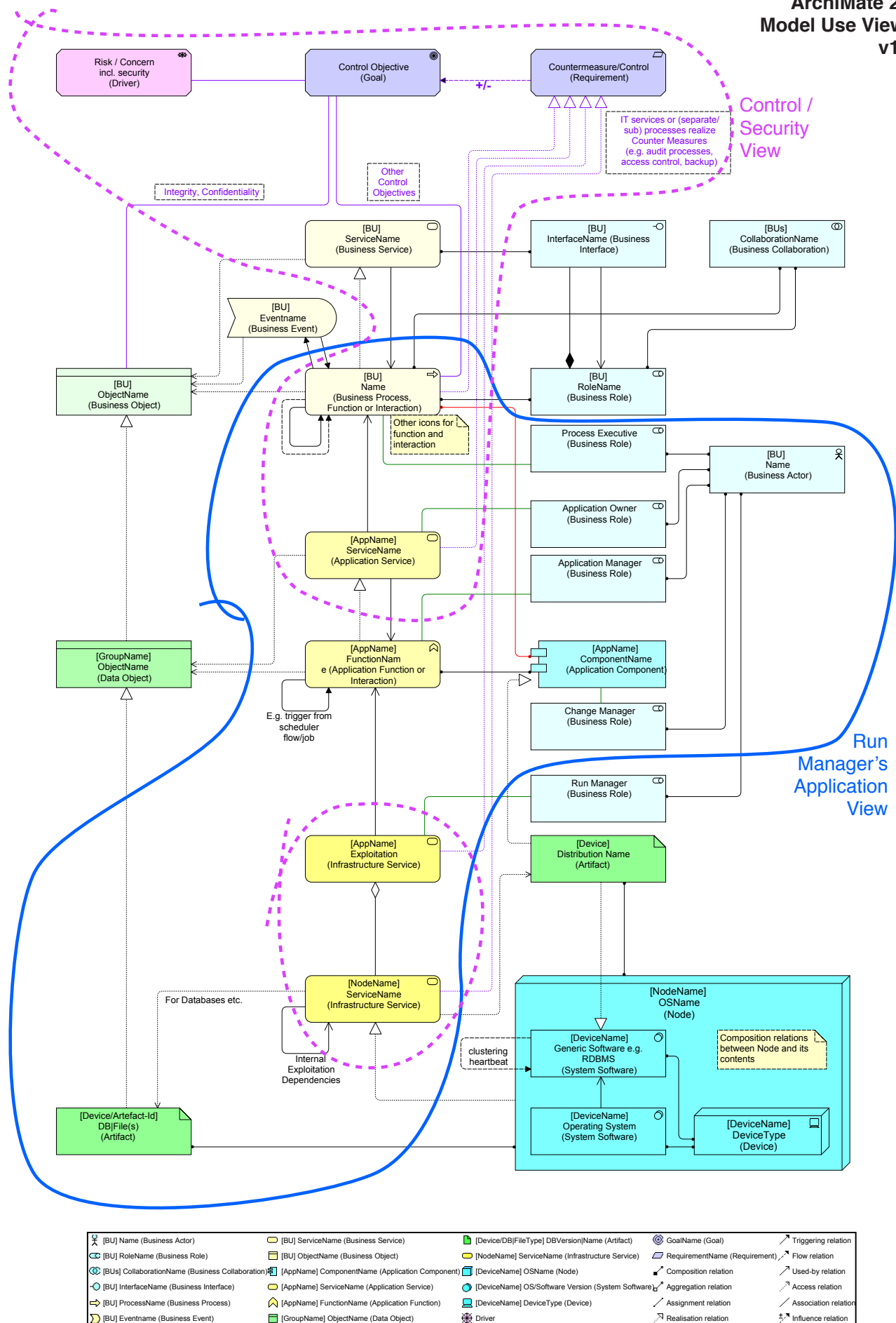
and Application Owner roles. You can see a version of the



View 149. Large Model Construction - Application Architecture View







View 151. ArchiMate 2.0 Use Views: Application Exploitation and Risk & Security



bit of a specific view about an application helps the IT Service Management department a lot to understand what it is they are actually managing. So he created an application context view for the Run Department which can be seen in View 151 on page 99 as the blue outline. I call this the 'Run Manager's Application View'. It can be seen separately in View 154.

Since the Run Manager already knows who he is and what his or her responsibilities are, he or she is not in the view. What is in the view is the basic application setup, which processes it supports and who the most important stakeholders are (Process Executive, Application Owner, Application Manager and Change Manager) from Section 12.9

## 16. Model Maintenance

A model, especially a large 'current state' model requires proper maintenance. For instance, you need to be kept up to date with changes. In our case, this for instance means that any change that the infrastructure people make in their tool is automatically mailed to the Enterprise Architecture department for maintenance. This must be so, because in our case, the CMDB is again fed from our ArchiMate model, so our model *has* to be correct. This feedback loop keeps the basic infrastructure information up to date. Here again, we meet Uncle Ludwig: through the link to the help desk, our Current State model is *used* and therefore acquires *meaning*.

For projects the situation is a bit different. Here the project first should have a project start architecture (PSA) that contains (the most relevant parts of) what the project is going to deliver. When the project finishes (generally, you can start with this when you are at the user acceptance testing phase, you make the model of the PSA complete and when the results of the project go 'live', you include the model of the PSA in your overall current state model.

When things change, you need a basic procedure for change. Here is mine:

- Views have a state:
  - \* *Work in Progress*
  - \* *Draft*
  - \* *Final*
  - \* *Erroneous*
  - \* *Research*

Normally, views progress in a cycle: they are created by an architect and during that phase they have the *Work in Progress* state. When the modeler is done, the view moves to the *Draft* state and will be checked. Generally, this must be done in two accounts:

- Is the view content-wise correct? You check that with people who can vouch for the correctness. E.g. business people for the business architecture, infrastructure managers for the infrastructure architecture, etc.

"Making it Practical: shortcuts" on page 87). Here, we do show the actual actors in our model. And we do not show the Counter Measures, because this is in informative contextual view about the application the Run Manager is responsible for to keep running.

Another view type you could add is a view for the Security Officer or the Operational Risk Officer. It is the light purple dashed outline in View 151 on page 99 and it can be seen separately in View 155. A view like this could show the Security or Operational Risk officer the most important objects related to Risk & Security management from the explanation in View 145 on page 93.

- Is the view properly modeled according to the chosen patterns and guidelines (e.g. style)? It is generally the lead architect who has to vouch for this.

When the view is checked on both accounts, it moves to the *Final* state. When something has to change again (e.g. an error has been found, a change must be made), the first thing to do is move the view again from *Final* to *Work in Progress*.

We also keep some views in an *Erroneous* state. These are known to be wrong but are currently not worked on. And we may have some *Research* views, which are generally temporary.

There is one caveat with all these views, and especially with the Construction & Use Views of the previous section. You can't have too many or maintenance becomes increasingly more difficult. Tooling (see Chapter "Tooling" on page 115) generally supports that if you change the label of an object in one view, it gets changed automatically in another (though label layout may suffer). If you remove a relation between object in one view, you might get warned it appears in other views. But if you add an object or relation that should appear in multiple views, you are on your own. Which means that the more views there are where a certain objects must appear, the more work changing a view becomes. Also, keeping a view easy on the eye (HVS, see Section 6 "Aesthetics (Style)" on page 39) sometimes means a lot of work. Add one little object and you sometimes have to layout 20 others and all their relations. Providing good readable views requires hand work (and hard work) and a good eye for layout, so prepare yourself. And don't expect a tool to be able to do this for you. Even the best tools in this area cannot produce the required cleanliness and organization of layout as View 77 on page 40 illustrates (to be fair, the changes in object sizes in that view were added by hand).

Finally: one important suggestion: Qlean Up! On a regular basis, check your model for objects that are unused (do not appear in any view and are also not children of objects that are used in a view. We use a script in our tool to do this. And of course, when IT gets decommissioned, make sure you remove it from your model. In our setup that means simply removing the views and then running the unused objects script.



# Architects Discussing



# Discussing ArchiMate

## 17. Proposed Improvements

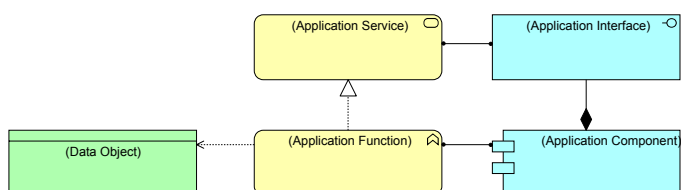
Overseeing the construction of a current state model of over 25,000 objects and relations has proven to me the usability of ArchiMate and has exposed a wealth about the reality of seriously using the language. In this section I'll make some proposals to improve the language. Now, while I of course prefer to see these changes implemented, I find the language without them still very powerful and usable for modeling Enterprise Architecture. It is just that our experience has been a very good test for the language and out of that test come these suggestions.

The first three suggestions are interrelated.

### 17.1 Switching Strength of Assignment and Realization

The strength order of structural relations was decided upon, but I have no documentation of why that particular order was chosen and privately I have been told it was more intuitively decided that reasoned.

What happens if we were to switch Assignment and Realization in the strength table for deriving structural relations? If we start in the middle of the ArchiMate 2.0 meta model, the basic Application Pattern (View 156):



View 156. The Basic Application Pattern

The *derived* relation of the route from Application Component via Application Function to Application Service changes from Realization to Assignment. Incidentally, that is the same result that we get if we follow the route from Application Component via Application Interface to Application Service.

This is kind of nice in two ways: first, because it does not matter which route you take what the derived result is. But secondly, because it is kind of nice to have Assignment as the resulting relation between an active component and a behavioral component. It means that you never break the

pattern that an active object is Assigned-To a behavioral object. Incidentally it also allows View 97 on page 60.

### 17.2 Service as Composite Part of Function/Process

The fact that an interface is a Composite of an active object while the service it is Assigned-to is Realized by the actor's function, could be changed as well. The argument could be made that the service — which is a 'visible/usable part of' the behavior it is Realized by — actually is a real 'part' and not something disjunct that is 'Realized'. If you describe that process of your business that provides a service, the description *will* include the interactions with the outside world. In fact, it has been originally defined that way by the ArchiMate designers. They wrote in the 1.0 specification:

*A business service is defined as the externally visible ("logical") functionality, which is meaningful to the environment*

In ArchiMate 2.0, this has changed to

*A business service is defined as a service that fulfills a business need for a customer (internal or external to the organization)."*

Apart from the circularity in this definition (a 'service' is a 'service', which does not explain what a service is) this is a nice change because it stresses the 'use' aspect of what a service is. To paraphrase uncle Ludwig: the meaning of service lies hidden in its correct use. And indeed, the explanatory text in the 2.0 specification says:

*A business service should provide a unit of functionality that is meaningful from the point of view of the environment.*

Here too, it seems that a service is a 'unit of' functionality, or in other words, a function of sorts: the 'usable function'. In an overview, all the service definitions from ArchiMate 1.0 & 2.0:

ArchiMate 1.0:

*A business service is defined as the externally visible ("logical") functionality, which is meaningful to*

the environment and is realized by business behavior (business process, business function, or business interaction).

An application service is defined as an externally visible unit of functionality, provided by one or more components, exposed through well-defined interfaces, and meaningful to the environment.

An infrastructure service is defined as an externally visible unit of functionality, provided by one or more nodes, exposed through well-defined interfaces, and meaningful to the environment.

ArchiMate 2.0:

A business service is defined as a service that fulfills a business need for a customer (internal or external to the organization).

An application service is defined as a service that exposes automated behavior.

The description adds: An application service [...] should provide a unit of functionality that is, in itself, useful to its users.

An infrastructure service is defined as an externally visible unit of functionality, provided by one or more nodes, exposed through well-defined interfaces, and meaningful to the environment.

As you can see, ArchiMate 2.0 kept the 'unit of functionality' phrase for Infrastructure Service in the definition. In fact, in the initial chapter on 'Core Concepts' of ArchiMate 2.0, a service is in general defined as:

A service is defined as a unit of functionality that a system exposes to its environment, while hiding internal operations, which provides a certain value (monetary or otherwise).

It adds:

[...] the service is the externally visible behavior of the providing system, from the perspective of systems that use that service;" and "The value provides the motivation for the service's existence.

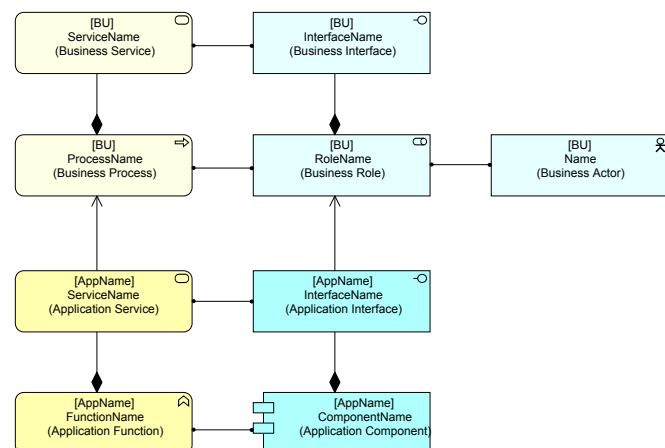
Given that we can follow uncle Ludwig in saying that meaning is hidden in correct use, we could give the following coherent definitions instead for the services:

- A business service is a usable part of business behavior. It may be used by business processes or functions (and by roles and actors);
- An application service is a usable part of an application function. It may be used by either business processes or functions or by other application functions (and by application components);
- An infrastructure service is a usable unit of an infrastructure function. It may be used by either application functions or by other infrastructure functions (and by Nodes).

Instead of 'value' I would prefer the more neutral term 'meaning' for the explanation of the service's existence. That nicely links with uncle Ludwig's relation between meaning and use. Value is then one of the types of meaning a service can have.

So, there are many pointers in ArchiMate (both 1.0 and 2.0) that a service is a 'unit of functionality' where the partition (and its meaning) is based on actual outside use. I suspect that in the behavioral column, it has been easy to talk about a service as being 'created' (realized) because the behavioral column is all about 'doing'. In the active column, it could only be sensibly seen as an interface being 'part of' an Application Component or Node or Role. But it works as well (even better) if we just see the service as a (usable) part of the function, just like the interface is a (usable) part of the role/component/node.

Changing the Realization relation (between function/process and the service it provides) to a Composition relation removes this unnecessary difference between the behavioral column and the active column. The result looks like View 157.



View 157. Proposal: A service is the Composite usable part of a Function

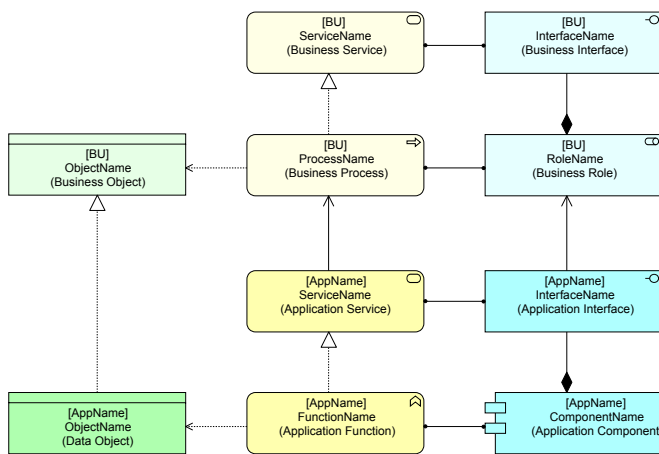
Incidentally, this has the same effect as the previous proposal: that the relation between an active object and its service is always Assigned-To, it does not depend anymore on the route taken.

## 17.3 Automated Processes

There are several problems with automated processes in ArchiMate, and they stem from the view the ArchiMate designers had on the Business Layer Architecture. They saw the Business Layer as all-human. This, by the way, is also a classical business process modeling approach, which then has the effect that automated processes end up being invisible in a business layer view of an organization.

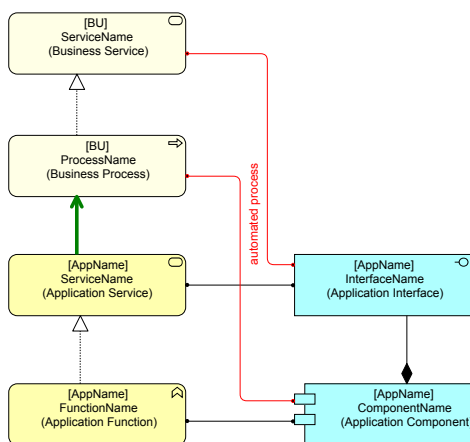
In ArchiMate, the human-oriented objects in the Business Layer use the Application Layer objects. Or, in the case of passive objects, the human-layer objects are realized by the Application Layer objects, as shown in View 158.





**View 158.** Human processes: Business Objects are Realized, applications are Used

So, when it became time to add automated Business Processes, they came up with the solution to have an Assigned-To relation between objects in the Application-Layer and the Business Layer. This way, the Application Component takes the position of the Business Role in non-automated processes. And the Application Interface takes the position of the Business Interface.



**View 159.** Automated Process: the Application Component is Assigned-To (performs) the Business Process and the Application Interface is Assigned-To (is the 'protocol' for) the Business Service

ArchiMate is not perfectly clear about how this should be used. Because there are two ways:

- Use *both* the Used-By (in green in the image) from Application Service to Business Process and the Assigned-To (in red) from Application Component to Business Process. Here you choose to keep the normal pattern (Application is Used-By the Business Process) and the Assigned-To is *extra*, to signal automation.
- Use *only* the Assigned-To from Application Component to Business Process. Here the Assigned-To comes *in place of* the Used-By.

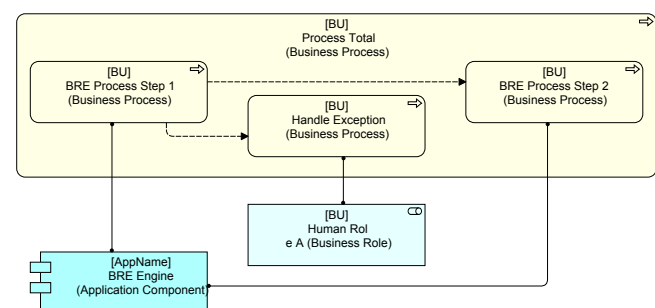
The second method has the following problem: though the Application Component Realizes a service for the business, you cannot model its use by a Business Process anymore, breaking the 'IT is used by the Business' pattern. If you leave out the Used-By that starts from the Application Service, there is not even a route from Application Service to Business Process that gives a valid derived relation. This

makes the Application Service a dead end, so why still model it?. But if you leave it out, the Application Function becomes a dead end instead, so why model it? And if you remove that, where is the behavior of the Application at the Application Layer? Because that might not be exactly the same as the Business Process. And besides, do you want an Application Layer behavior description in your Business Process? Something you discuss with the Application Architects?

The first method had another problem in ArchiMate 1.0 that was fixed in ArchiMate 2.0: Technically, you had modeled that the Application Component used itself: Application Component Assigned-To Application Function Realizes Application Service Used-By Business Process Assigned-To Application Component, derived relation: Used-By. But the one-way nature of Assignment in ArchiMate 2.0 fixes that.

The Assigned-To relations from Application Layer to Business Layer are therefore rather messy shortcuts. A bit like the missing Infrastructure Function and the resulting behavioral status of the System Software object in ArchiMate 1.0 that were repaired in ArchiMate 2.0.

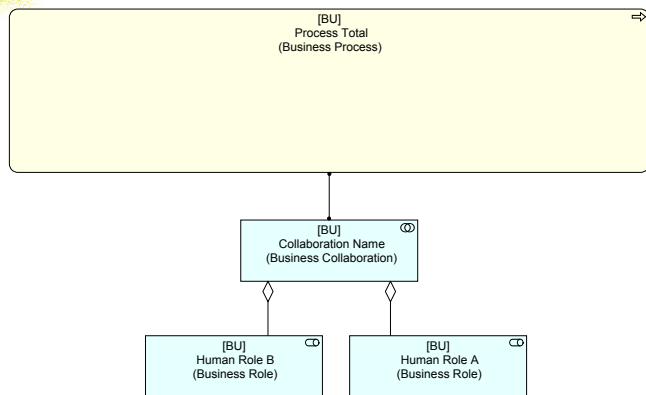
There is another related problem. Suppose you have an automated workflow in a Business Rule Engine. Most of your Business Process runs automated. But once in a while, an exception happens and a human needs to do something. This is quite a common pattern and ArchiMate does have problems with it. View 160 shows how it basically looks:



**View 160.** An automated process with human exception handling

Now suppose I do not want to show the internal details of 'Process Total'. The cleanest way to do this is to leave out the subprocesses, make the roles Aggregate parts of a

Collaboration and Assign the Collaboration to the process. In a fully human world, it looks like View 161.



**View 161.** When two roles separately do the (sub) Business Processes, the overall Business Process is performed by a Collaboration

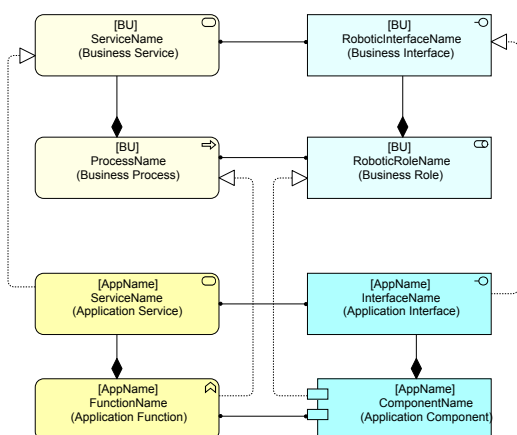
**Question 7.** Can you spot the ArchiMate error in View 161?

**Answer 7.** You're only allowed to Assign an Interaction to a Collaboration

But with our *automated* process as one of the parts of the overall process, we cannot do that. An Application Component is not allowed as a child of a Business Collaboration in ArchiMate. In fact, when we have automated processes, ArchiMate does not allow us to use Business Layer objects to model an active business side of the architecture properly at all. So modeling processes that are *partly* automated becomes problematic.

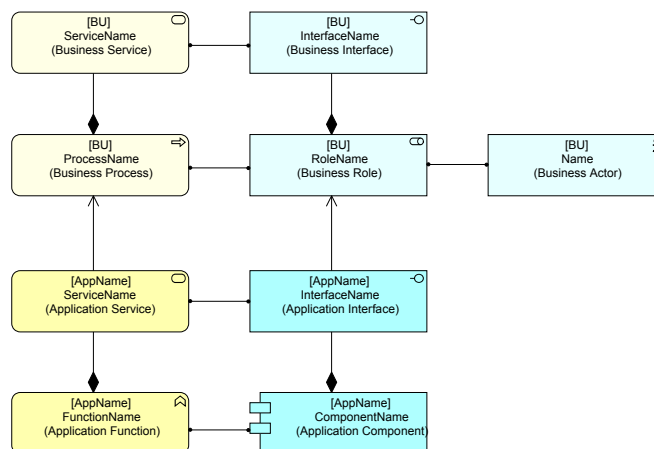
There is a straightforward way to solve this. We can look at the automated process as being performed by a robot object of sorts at the business level. There are various ways to implement this, but my preferred solution is to have an Application Component *Realize* a Business Role (Realizing a Business Actor leads to unnecessary complexity, I will not go into that here). To make the whole model simpler, I also make the relation between a process/function and its service to be the same as between the actor and the interface, namely Composition, as proposed in 17.2 "Service as Composite Part of Function/Process" on page 103. That is not strictly necessary for my argument, but I think that also could be an improvement for ArchiMate.

It looks like View 162 for an automated process:



**View 162.** Proposed meta-model Realization relations for automated processes

And it looks like View 163 for a non-automated process:



**View 163.** Proposed solution for non-automated processes

In my opinion, this cleans up and improves ArchiMate in a few ways. First, the types of relations between layers have been reduced to Realization and Used-By. A lower layer active/behavioral object is Used-By a higher layer, or a lower layer passive object Realizes a higher layer object. A direct Assigned-To has become limited to layer-internal relations.

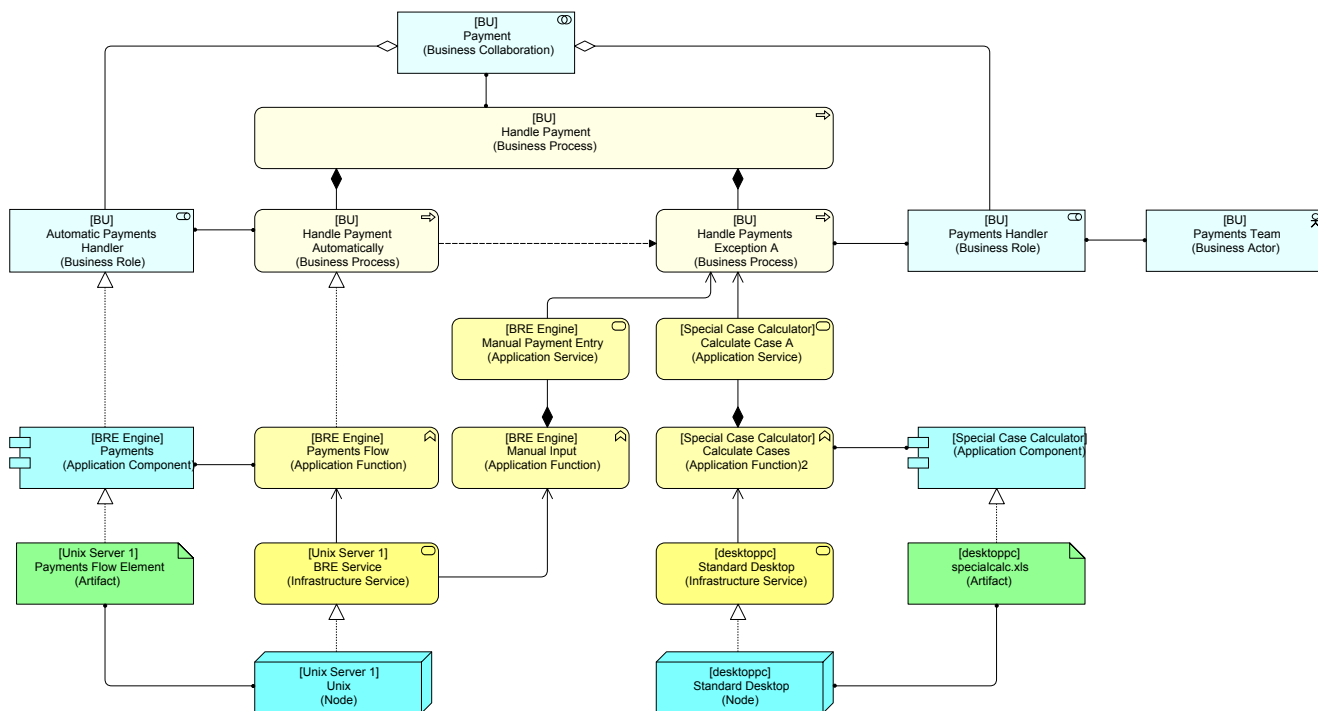
Secondly, we mirror the relation between the actor and the interface to the one between the actor's behavior and the service: composition. If you think about it, it is unnecessary (as is the case in ArchiMate 2.0 and 1.0), that the 'visible part of the behavior' (or maybe better: the 'usable' part of the behavior), the service, is disjunct from 'the behavior'. Take an Application Component: it is Assigned-To its behavior (the Application Function), but not to its 'visible' behavior, because the derived relation via Application Function is Realization. It is simpler and cleaner to look at the 'visible' behavior as part of the 'overall' behavior. This is also more in line with ArchiMate's fundamental divide between its actors and their behavior.

When we add the possibility for an Application Component to Realize a Business Role (and for the Application Interface to Realize a Business Interface) like we have done here, the older cross-layer Assignment relations have gone. The derived relation between Application Component and the automated Business Process is Realization, and this is true too for the derived relation between Application Interface and Business Service. This breaks the clean use of Assignment between active and behavioral components and it is so because Assignment is a stronger structural relation than Realization. But if we change the strength order as proposed in 17.1 "Switching Strength of Assignment and Realization" on page 103, the old layer-crossing assignment reappears as a valid derived relation again.

## 17.4 Summary of the previous three change proposals

The three proposals on the previous pages were:

- Change the strength order of Assignment and Realization;
- Let a service be the Composite part of the function/process that is responsible for it;



**View 169.** An example that uses the proposed changes to ArchiMate from sections 17.1, 17.2 and 17.3. A Business Process is performed by a Collaboration of humans and applications. The process is automated using a BRE, humans handling the exceptions, using a user interface of the BRE and some hand work.

- Make sure automated processes become properly represented in the Business Layer Architecture, by making an Application Component able to *Realize* a Business Role;

If we would implement all three proposals, the effect would be:

- We can correctly make collaborations of automated and non-automated business processes (e.g. automated business processes with human exception handling);
- We are ready for a future in which more and more IT agents will be recognizable as actors at the business level (e.g. automated phone services or interactive web sites of today, but then on steroids);
- Only Realization and Used-By are possible between architecture layers. This is both intuitive and more simple than in ArchiMate 1.0 & 2.0. Also, Realization is only possible *between* layers, making it a pure layer-abstraction relation.
- The (derived) relation between an actor and the behavior it is responsible for becomes Assigned-To *in all cases*. This is a more proper derived relationship between a structural component and a behavioral component than Realization.
- The old layer-crossing Assignment relation is saved as a derived relation, which makes the set of changes backwards compatible.

Using Composition between a function/process and the service it provides/contains also clears up an important point of style mentioned in a previous chapter: a service can in ArchiMate 1.0 & 2.0 be Realized by multiple processes/functions. But such a construct is problematic at the application level, because it is anthropomorphic (Section 8.2 "Application Collaboration is an Anthropomorphism"

on page 63) and generally not realistic in IT terms. At the business layer, such sharing *is* meaningful, and it can still be modeled: through Collaboration.

We can top this off with an example in View 169. There you'll see how a Business Process that is automated using a Business Rule Engine has a manual part that handles exceptions. Here, the human uses a spreadsheet on his desktop to calculate something and then enters it manually into the BRE-based system. Note, the actual ruleset artifacts and such of the BRE have been left out here. They can be seen in the basic BRE Pattern in View 85 on page 50.

## 17.5 Have only Business Service in the Product Aggregate?

The Product Object in ArchiMate can Aggregate Contracts, Business Services, Application Services and Infrastructure Services. First, this is a combination of passive and behavioral objects and as such the 'passive' status of Product is circumspect. But on the other hand, it is a nice grouping of what businesses do, why they exist at all.

However, suppose we adopt the proposals of the previous sections on automated processes (and thus automated services). We are then able to say that a fully automated Business Service is Realized by the Application Service (View 162). So, a Product would never explicitly have to Aggregate the Application Service, it should just Aggregate the Business Services, including those Realized (automatically) by an Application Service.

Actually, I am not quite convinced that simplifying a Product to only Aggregating business layer objects is a good thing. There is too much abstraction and too little reality. So, I would stick to just adding Infrastructure Service to the objects that a Product can Aggregate.



## 17.6 Actors Aggregate Roles?

A Business Role in ArchiMate 1.0 was:

*A business role is defined as a named specific behavior of a business actor participating in a particular context.*

Role — while being a *structural* object — was defined in *behavioral* terms. In ArchiMate 2.0, it has become:

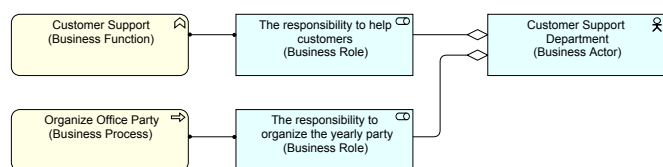
*A business role is defined as the responsibility for performing specific behavior, to which an actor can be assigned.*

My proposed new definition after adding the possibility to let an application Component Realize a Business Role as proposed in section 17.3 on page 104, would simply be:

- A business role is defined as the responsibility of performing specific behavior.

and added to that in the description would be: “A business role can be fulfilled by an actor or realized by an application component”.

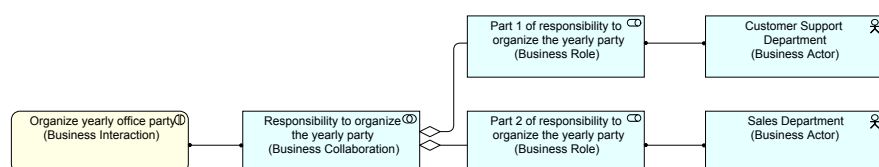
The change from ArchiMate 1.0 to ArchiMate 2.0 has removed the behavioral definition and replaced it by the term ‘responsibility’. Now, a person ‘has’ a responsibility, so why not model it that way? And since the responsibility can be shared, and since it can exist without someone being responsible (it happens in real enterprises), the obvious relation to use is Aggregation. That also removes the situation that Assigned-To is used for two different meanings. Between structural and behavioral object it means ‘performs’, but between Business Actor and Business Role (both structural objects) it means ‘fulfills’ (and now ‘has’ responsibility). It also removes the somewhat ugly two-step of a double Assignment to go from Business Actor to its behavior (e.g. Business Process) and it more clearly cements the structural status of Business Role which after all can be seen as a sort of ‘virtual actor’ that can perform behavior like a Business Process. An example is in View 164.



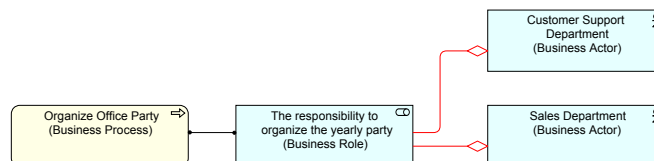
**View 164.** A Business Role as Aggregate child of a Business Actor

In this example, the Customer Support Department has two responsibilities: the responsibility (Role) to support customers and the responsibility (Role) to organize the yearly office party.

But what if two Departments would both have the Role to organize the office party? We could model it like View 165.



**View 170.** Customer Support and Sales Work together to organize the yearly office party



**View 165.** Customer Support and Sales both have the responsibility (Role) for organizing the yearly office party

But the question is: are they *separately* responsible or do they have to organize the party *together*? The answer is, separately, because if they *have* to organize it together they apparently play *different* roles, and in that case we should use a Collaboration as in View 170. (And for good measure, have another look at View 22 on page 24 and what was said there).

This change would not make me use Aggregation as well for the relation between an Application Component and the Business Role it Realizes (as proposed earlier in section 17.3, View 162 on page 106). In fact, I kind of like the difference as it illustrates the free will we assume for the human actors (who are able to drop that responsibility), while the Application Component has no choice: it cannot but Realize the Business Role. ‘Actor’ and Role are fused into one. The consequence of this reasoning is: if we ever will be able to create robots that we think of as having free will, we will have to model them as Business Actors and give them rights, the right to strike for instance...

Note: we could consider other options, like having Business Actors Realize Business Roles just as proposed for Application Components in Section 17.3.

## 17.7 Only use Assignment for performed-by

The Assignment relation is pretty heterogeneous. It covers three different meanings:

- Between a Business Role and business behavior and between Application Component / Interface and Application Function / Service it means *performed-by*;
- Between a Business Role and a Business Actor it means *fulfilled-by*;
- Between an Artifact and a Node it means *deployed/resides-on*.

In ArchiMate 1.0, the relation was bidirectional, in ArchiMate 2.0 it has become unidirectional. Clearly, the wide scope (between active and active, between active and behavioral and between active and passive, skipping the behavioral column altogether) of the Assignment relation may confuse people.

As I wrote in the previous section, one of these confusions can be easily removed: if we say that a Business Actor Aggregates Business Roles, we lose one of three meanings.

The question is what to do with the meaning that resides at the infrastructure level. The designers of ArchiMate tried very hard to minimize the number of concepts and relations and having a different relation for ‘resides-on’ goes against that goal. But

without the Assignment here, Assignment would be nicely restricted between active and behavioral elements, which also is a form of cleanliness.

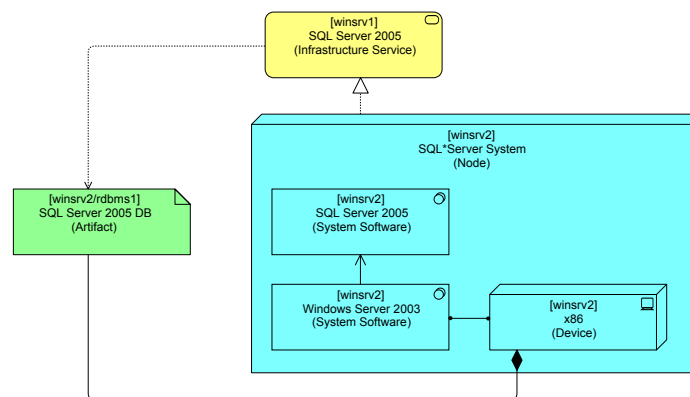
We could do without this Assignment, possibly. In ArchiMate 2.0 we have the Assignment relation from the Node to the Infrastructure Function and the Access relation from Infrastructure Function to the Artifact. Together these produce the derived relation Access. But the problem is that an Infrastructure Function may Access an Artifact that resides on another Node altogether, so this does not help us.

But if we look a bit deeper with what we mean with resides, we find an alternative. For an Artifact to reside on a Node, its bits must be physically located on a Device. In fact, *the bits of the Artifact are parts of that Device* in the real world. So, alternatively, we could say that an Artifact is a (Composite) part of a Device. And via the derived relation Node via Composite to Device via Composite to Artifact, the Artifact will also be a Composite part of the Node. This fits nicely with the underlying idea in ArchiMate that an Artifact is a physical manifestation, as is the Device.

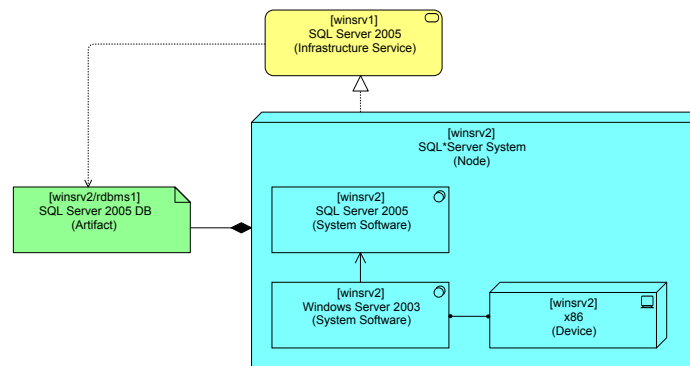
In summary, we could simplify the use of Assignment to a relation between active and behavioral components *only* and it always means *performs*. The Assignment between Actor and Role (fulfills) could become an Aggregation and the Assignment between Node and Artifact (resides) could become a Composition between Device and Artifact instead (View 171).

And since we can make a Device a Composite part of a Node (as in our pattern in View 57 on page 46) we have a derived relation from the Node to the Artifact: Composition and Composition becomes Composition (View 172).

There is a disadvantage of this approach. In the real ArchiMate 2.0 meta-model, the derived relation from Node, via Assigned-To to Artifact, via Realization to Data Object and via Realization to Business Object is Realization. This does not change when the Assigned-To from Node to artifact becomes Composition, as suggested. But if we would leave it at Assignment and follow the suggestion of section 17.1 “Switching Strength of Assignment and Realization” on page 103, the resulting relation would still be Assigned-To in its meaning of ‘resides on’. In other words, follow the earlier suggestion and not this one, and the Node is Assigned-To the Business Object, or, the Business Object resides on the Node. That can be useful too. Some more analysis is needed.



View 171. Artifact as Composite part of Device



View 172. Artifact as Composite part (derived relation) of a Node

## 17.8 Allow multiple parents in a Composition

In ArchiMate, it is stated that an object can only be part of one composition. It has taken that from UML, the modeling language for software engineering.

Now, in software engineering, composition and aggregation are (also) about memory management. Who owns what piece of memory? Who should free the allocated memory? When an object is a composite child of another object, if the parent is deleted, the child should be deleted also. However, when an object is an aggregate child of another object, and the parent is deleted, the child should be left alone. An aggregation only *points* to another object, it does not *own*

it. In software engineering, accessing freed objects used (e.g. as a result of ‘memory sharing’) to be a major source for crashing applications and not freeing objects used to be a major source for ‘memory leaks’. Hence, ‘memory sharing’ and ‘memory leaks’ became a sign of bad programming in the early days and the paradigm has stuck, has become a foundational part of OO and UML and has been transported into ArchiMate.

But relaxing this for Enterprise Architecture might be very useful as we *do* have real and unavoidable sharing. Take the following example. Suppose we model our Infrastructure Services for the exploitation of applications as being composed of several parts, which we name ‘infrastructure building blocks’ (see 7.6 “Infrastructure ‘Building Blocks’” on page 50). For instance, suppose an application requires a file share, a relational database and a system where the application is executed. In a previous chapter, we modeled this by creating a specific abstract Aggregation of Infrastructure Services, that together form the Aggregated ‘Exploitation’ Infrastructure Service that provides TI-support for the application. An example can be seen in View 173 on page 110.

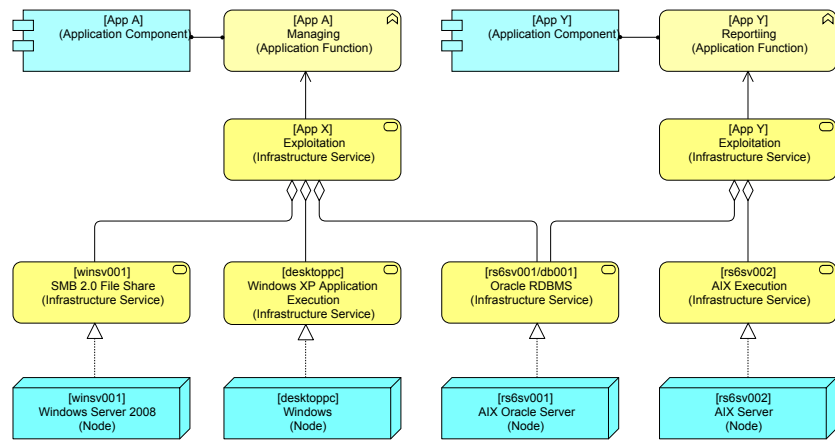
This aggregation breaks the possibility to have derived Used-By relations between the individual building blocks and the applications that depend on them. Using derived relations, we cannot say, “[rs6sv001/db001] Oracle RDBMS (Infrastructure Service” is Used-By “[App Y] Reporting (Application Function)”.

But what if we were to release that restriction on multiple parents in a Composition? Suppose we would allow true sharing as in the nasty ‘memory sharing’ of the early programming years? We could model it like in View 174 on page 110.

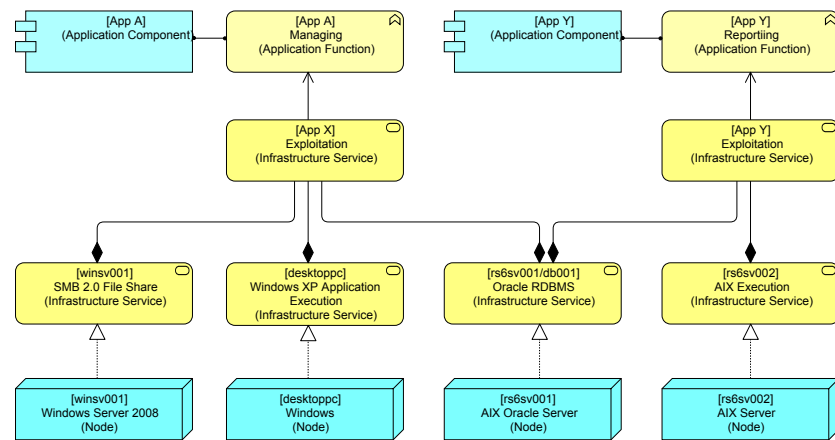
In the Enterprise, true sharing is not only real, it is often even a desirable state of affairs, cost-wise. Now, if we would allow multiple parents in a Composition in ArchiMate, we could create real sharing in our models. Remember, if one of the now 'parent' Infrastructure Services breaks, the 'child' Exploitation Infrastructure Service, which in the case of App A in View 174 has three 'parents', breaks too. Which is as it should be. Now we can say: "[rs6sv001/db001] Oracle RDBMS (Infrastructure Service" is Used-By "[App Y] Reporting (Application Function)". So the advantage for this pattern is that the official derived relations still can be used.

There is however also a disadvantage. The aggregation version is easier to read for people and the use of the symbol becomes different from UML and that means another addition to the learning curve (this time for software architects). Multiple parents confuse. So, personally I can live with the aggregation version (the derived relations are too limited for much analysis anyway) and the fact that it does not clearly shows the real 'make or break' dependency of the abstract 'Exploitation' Infrastructure Service on all its building block components.

All in all, relaxing the single-parent limitation makes sense because true sharing is a desirable state of affairs for Enterprise Architecture and it solves a derived relations problem in ArchiMate, but I am unsure if I would use it myself.



**View 173.** *Exploitation Infrastructure Services based on Aggregation of (shared) Infrastructure Service Building Blocks*



**View 174.** *Exploitation Infrastructure Service based being a Composition child of Multiple parents*

## 17.9 Infrastructure Collaboration and Interaction to model Clustering

There might be a structural way to support the modeling of various ways of clustering (instead of using dynamic relations like flows for this, as in the patterns of for instance 7.19 "Infrastructure Pattern: High-Available Database Cluster" on page 60) in the infrastructure layer if the infrastructure layer would support its own Collaboration and Interaction objects.

### 17.10 Clean Up the Location Concept, possibly change Grouping

ArchiMate 2.0 introduced the Location object. But there are a few things amiss with the specification:

- The definition says it is to be connected to an *active* object and 'indirectly' to a behavioral object.
  - \* It is unclear what 'indirectly' means.
  - \* In Figure 9: Business Layer Metamodel, Location is connected to Business Actor, but also to the *passive* objects Business Object and Representation.
- It is unclear what a Location of a Business Object (e.g. "Bank Account") can be other than as derived relation via Artifact (and if it is a derived relation, it should not be part of the core meta-model).

The example in the specification would probably have been modeled as Grouping in ArchiMate 1.0. But Grouping is not an object, it is a relation. Instead of creating a new object for every use (now Location, in the future maybe something else), Grouping could have become a true object (a Group). A Group is then a generic object. As all concepts can Associate to each other, a Group can associate to all other concepts. We then lose one relation type (that does not look like a relation and does more or less the same as Association) and win one concept type (Group). Since we then have Group, we can either lose Location (can be done with a Group) or make it a Specialization of the Group concept that uses Assignment instead of Association to link to its constituents.

This is all not very important for the usability of ArchiMate (though I think it cleans it up). But it does trigger me to propose another change:

### 17.11 Remove Nesting as a Language Concept

Nesting, in the current specification, is mentioned as a way to represent the Composition, Aggregation and Assignment relations. And (as mentioned above) it is used to represent the Association relation in the Location concept example.

I think, Nesting, as a concept could be removed from the grammar and made a purely visual construct *that hides relations* (and hence is a bit dangerous).



You could define Nesting as a way to hide relations between objects and force objects to have at least one relation if they are to be nested. Making it a purely visual construct would also allow views where an object is part of two different Nestings (e.g. as in View 125 on page 75)

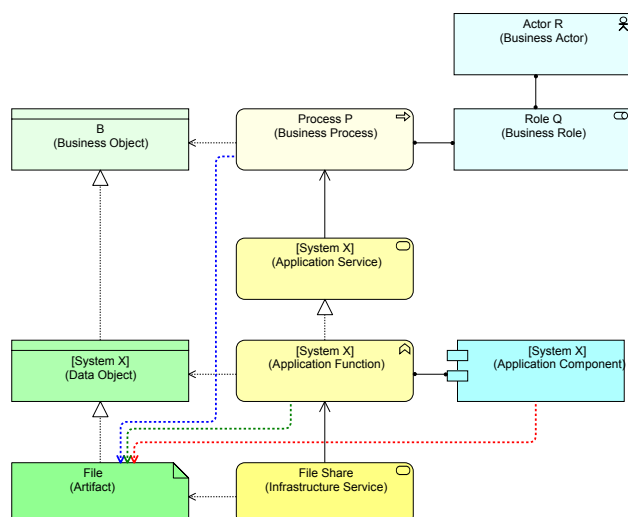
## 17.12 Make the Access relation bidirectional

ArchiMate 2.0 removed the bidirectionality of the Assignment relation, and that was a good move. The bidirectionality led to all kinds of senseless derived relations. ArchiMate 2.0 removed all of those and added the ones that made sense 'by hand' to the core meta-model.

So, why propose now to make another relation bidirectional? Well, what drives this is a couple of realities, like:

- Some processes are just about loading an Artifact in a system. You can imagine Data and Business Objects, but the labels you give these are just representations of the Artifact. They add to the complexity of the model, they do not add to the informational content of the model.
- Behavior may depend on passive objects. A good example is application maintenance from 12.6 "Secondary Architecture: Application Maintenance" on page 86. Here, the application maintenance process edits a file, say an ini file, that influences an application's functionality. The application's functionality is dependent on the settings in the file (on the 'Settings' Data Object the Artifact realizes). Though the Artifact is shared, the Data Object isn't, which shows up when you make errors in that ini and the application crashes.

To illustrate what derived relations we can have when Access becomes bidirectional, have a look at the green and blue Access relations in View 166.



**View 166.** Three useful (derived) Access relations

The green one follows from Artifact is Accessed-By the Infrastructure Service, which is Used-By the Application Function. The result is Artifact is Accessed-By the Application Function. The blue one follows the derivation up to the Business Process.

Enabling this, frees you from creating 'unnecessary' Data Objects and Business Objects. Yes, they can be imagined, but the result makes your model contain abstractions that have a negative effect on understandability.

As Assignment is (rightly) unidirectional, Access relations from higher layer active components must be added 'by hand' to the core model, like the red one.

## 17.13 Get rid of Interactions

When we discussed Business Functions and Business Processes (Section 10.2 "Business Function or Business Process?" on page 72) and Business Process Modeling (Section 10.4 "The 'End-to-end' Business Process" on page 75) we ended up using Business Collaborations, but we (illegally) assigned them to Business Functions or Business Processes instead of Business Interactions.

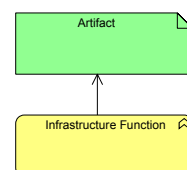
But these Business Interactions were in fact processes or functions that were performed by a collaboration of roles. And then you have to ask yourself: what is an Interaction? Is it a type of process (defined 'outside-in' on the basis of what it produces) or a type of function (defined 'inside-out' on the basis of what it needs)? An Interaction can be needed in either situation.

If you think about it, the way a Collaboration is Assigned-To a Business Process (as in View 118 "A Business Process using Business Functions is performed by a (loose) Business Collaboration" on page 74) makes a lot of sense. And that end-to-end process is in fact performed by a collaboration. Technically, I can Assign the Collaboration to that process, but ArchiMate says in its definition I'm not allowed to: a collaboration should only be assigned to an Interaction.

There is no need for this special status of interactions. If you keep collaborations and remove interactions, ArchiMate becomes simpler and sense-making models like that one from View 118 become legal.

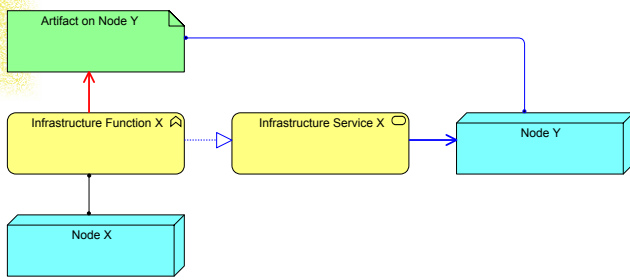
## 17.14 Get rid of unwanted derived relations explicitly

I once saw View 167 modeled.



**View 167.** An Infrastructure Service Used-By an Artifact?

This was created by a tool that implements ArchiMate and stops you from making unallowed relations. So, I was curious, how come ArchiMate allows a passive object to use something? It completely makes no sense. I came up with the blue route in View 168:



**View 168.** How the Infrastructure Function can be (Used-By (derived) by a passive Artifact in ArchiMate

Node X is there for illustration that there are two Nodes in the background of pattern View 167. ArchiMate 2.0 removed the bidirectionality of Assigned-To. Then it added the explicit Used-By (the blue one) from Infrastructure Service to Node to make it again possible that a Node Uses an Infrastructure Service. While that makes sense, this enables the route to go on using an Assignment from a Node to an Artifact with the senseless red Used-By as a consequence.

This can quite easily be fixed. Just as one can add needed relations by hand that cannot be derived, we could prune the derived relations that make no sense. It's a pragmatic solution for an unwanted consequence. It's not a big deal, I think, that we have to admit that the logical procedure for creating derived relations does not — by definition — results in a relation that makes sense. Uncle Ludwig would not be surprised.

## 17.15 Get rid of Meaning

There are a few reasons to get rid of meaning. First, the specification is incoherent about it (e.g. how it is described in the Motivation Extension versus in the Core). Second, it has an overlap with Goal. Third, it runs foul of Uncle Ludwig.

## 17.16 Make a Motivational Element in ArchiMate's Motivation Extension a Specialization of Business Object

ArchiMate's Motivational Elements (Driver, Assessment, Goal, Principle, Requirement, Constraint) can be easily seen as types of Business Objects. The easiest is Assessment. An Assessment has to be made by a process and it can be Accessed-By by a process. But you can say the same about all the other Motivational Elements.

Doing this makes it easier to link the secondary and tertiary processes (See 12 "Secondary and Tertiary Architecture" on page 84) like Operational Risk Management and Continuity Management in your enterprise to your primary process.

## 17.17 Give Product a non-passive status

A Product (see 2.4 "Products, Contracts and Value" on page 25) is made up of both passive and behavioral objects. The Product concept itself is classified as passive, but at the same time it may Access Business Objects as it were behavioral.

Basically, Product is a useful object, but it doesn't really fit in ArchiMate's core construction very well. I would suggest creating a specific category for Product, for instance "Collection Concepts". The object-type Grouping as proposed in Section 17.10 on page 110 could be one and Location (currently classified as active, which also does not make a lot of sense).

Any object of the Collection concept type could be associated with other objects.

## 17.18 Improve the description of Business Function

The specification's description of Business Function stresses the classical picture of the use of Business Functions by end-to-end processes too much and ignores how a Business Function in reality Realizes a service (e.g. by an *internal* Business Process as described in 10.2 "Business Function or Business Process?" on page 72. The description of the complex relation between Business Functions and Business Processes could be improved, I think (that is, if the ArchiMate Forum's members agree with my suggestion for their use).

## 17.19 Are the proposed improvements really necessary?

I think ArchiMate is great. It is the best thing since sliced bread (to be honest, I prefer the taste of freshly baked bread I slice myself, but I digress) for Enterprise Architecture modeling. The language is not strictly formal, but its concepts and relations have been selected for usability, and as uncle Ludwig explained to us, that it one of the best tests of meaningfulness.

The world of Enterprise Architecture stretches from the strictly logical world of bit and bytes to the not-always-so-logical world of human behavior. It is unavoidable that such a stretching exercise leaves its marks. So, it is easy to find (logical) fault with the language here and there. But from a business perspective (a human perspective) it is very good at enabling you to model to the extremes of Enterprise Architecture. And the fact that — even without all these improvements — we still were able to use the grammar to the extent we did shows how powerful the language in fact is.

With the right use of patterns and the right discipline and a good knowledge of the powerful underlying ideas, you can take this language far. Even without the improvements proposed by me in this chapter.





gniloot

O

!

U

oo



# Tooling

## 18. Multiple Models of One Reality

You can create your views in a decent drawing tool, like OmniGraffle Professional for the Mac or Visio for Windows. For both applications, so-called ‘stencils’ are available, and you can use these to create views. In fact, you can use any drawing tool, the stencils and the smart behavior of the graphics only make life a bit easier. But the main problem of tools like these is that they generally are just that: a *drawing* tool, they create views but they do not create a *model*. What you really need is a good modeling tool.

Now, it is important to realize a very important aspect. *There will by definition be multiple models of the same reality in your organization.* There is no single model-based tool that supports all uses of any model of your enterprise. The help desk needs a model of all the infrastructure and applications that run on it and if they are professionals, they want to know what business processes are supported. They might want to add aspects like application owners, process owners and such. All this you can do in ArchiMate. But their model needs more: they need to log incidents against applications, they need incident management work flow support and case management, etc.. The operational risk managers need a model with risks, control objectives, control measures and also the business processes, functions, roles and actors involved. ArchiMate can do that. But they also need to store assessments, have work flow support for incident management, require strict access to maintained data, etc. The Business Continuity Managers need Continuity Plans with business processes and the Business Control people need detailed process descriptions with more detail than what is in your Current State model. The list is long and all these different uses require often subtly or not-so-subtly different IT support. And there will not be a single tool that is going to support them all. Your Enterprise Architecture modeling tool is not the tool to log incidents, store improvement plans, etc.

The consequence of this is that it is very important to realize that you will always, by definition, have multiple models of your Enterprise reality in your organization. And these different models even need to be used in conjunction sometimes, for instance when auditors check how well prepared you are for calamities, and they want to look at your Security Architecture, your help desk setup and your Risk Management. If these three systems have an incompatible description of, for instance, your Business Processes, Business Functions, Roles, IT Services, etc., it will be impossible to get a good look. So what generally happens is that the people helping the auditors, create their own ‘model’ of the organization with everything that specific auditor need. What they create is often incompatible with what has elsewhere in the organization been

documented and it will certainly be a duplicate or triplicate effort with another maintenance burden you do not want. You don’t want that state of affairs.

Since you have to start from the assumption that it is unavoidable that there will be multiple models needed of your reality, the question does not become: “How do I create *the* best model?”, but “How do I make sure the *different* models tell a *single* story?” and “How do I prevent a duplicate of effort?”.

There are two ways to make that happen:

- Have one model be the slave of another (master) model. E.g. you export your Enterprise Architecture model from your EA modeling tool and you import it in another tool.
- Make sure all models are compatible enough so they can be translated into each other or reconciled against each other.

Which you use and — in case of the first approach — who is master and who is slave, how does synchronization work, is something that you need to design carefully.

For instance, where I work we use our EA Current State model effectively as our CMDB. Our EA tool has a decent scripting language and import/export facilities. We export our model (the master) in a way (spreadsheets) that it can be imported in our IT Service Management system (the slave). Projects that go live are added to our Current State model and thus exported to the ITSM system. But during day-to-day operations, small changes are documented in our ITSM system. The system automatically sends these to a mailbox of Enterprise Architecture and we add the changes to our model. New items get an id in the ITSM system and this id is fed back into our EA model as a property of certain objects. If we change something in the Current State model and export to the ITSM system, we can change names of objects, and the id makes sure the new names are adopted by the ITSM system. Once in a while, we run a reconciliation to find any item that has slipped through our net. It is difficult to get this right, but it is doable.

For other systems (the software engineer’s design system, the operational risk manager’s risk management system, the process modeler’s process modeling tool) we must make sure that their model is compatible with the Enterprise Architecture model, which plays the role of a central core repository.

## 19. Tool Aspects that may be important

---

Depending on your requirements, different tools may fit your bill. If all you want is directed drawing with not too complex views, many tools will do what you want.

From a perspective of very large models, the following aspects are important:

- Does the tool support everything in ArchiMate?
- Is the tool not overly restrictive?
- Can the tool handle large complex models and views without crashing, slowing down, etc.?
- Does the tool support a modeling style that is usable for large complex models and views?
- Is it possible to configure/use the tool such that it supports your preferred style?
- How good is the support for drawing tricks (e.g. layers, etc.)?
- Can you influence layout (colors, standard labels, attachment points for relations, ordering of relations, etc.)?
- Does the tool have a scripting language?
- Can objects and relations be augmented with properties?
- Does the tool have good import/export facilities?
- Can the tool produce or make available reports for people that do not use the tool themselves? Do these reports work when
  - \* Views are large and complex (e.g. it is not scaled such that the contents become unreadable)
  - \* Some views should be reported but others not (e.g. do not report work in progress views)
- Can the tool produce vector-based output (e.g. PostScript, SVG or — preferably — PDF), e.g. for large poster prints?
- Is the tool well supported?
- Is there an active and experienced user community for the tool?
- Is the tool's file format open for inspection (and emergency repairs)?
- Is it possible to anonymize models? This is useful if you want to send models to tool support or forums without having to disclose company sensitive information.

## 20. Tool Reviews

---

I am still in doubt about providing tool reviews here. Maybe in a second edition if the book is successful. I have identified the following applications and suites that claim to support ArchiMate modeling (I have excluded one that also claims it but that I know from first hand experience cannot deliver):

- Archi
- ABACUS Avolution

- BiZZdesign Architect
- Software AG ARIS
- Sparx Enterprise Architect

The diagrams in this book have been made with BiZZdesign Architect. I am not affiliated with, nor do I endorse BiZZdesign Architect or any other tool at this point in time.



This page intentionally not left blank

# Index



# Index

## A

Access relation 16  
Aesthetics 39–44  
Aggregation relation 20, 24  
Application Collaboration 24, 63  
Application Component 15  
Application & Data Layer 10  
Application Function 15  
Application Interaction 24  
Application Interface 16  
Application Service 16  
Application Using Another Application 17  
ArchiMate  
    Motivation Extension 91  
    style 39  
    using color in 42  
    using label text in 43  
Artifact 19  
Artifacts resides on the Node 19  
Assignment relation 16  
Association relation 24  
Automated Processes 26

## B

Basic Application Pattern 15  
Basic Business Pattern 16  
Behavior versus Structure 33, 71  
BITMAP 83  
Business Actor 18  
Business Collaboration 23  
Business Event 22  
Business Function 18  
    dividing your enterprise into 77  
    multiple functional landscapes 81  
    requires internal process to provide a service 73  
Business & Information Layer 10  
Business Interaction 23

Business Interface 17  
Business Object 17  
Business Process 16  
Business Role 16  
Business Service 17

## C

Change Architecture 11  
Communication Path 26  
Composition relation 16, 20, 24  
Contact the Author 12  
Contract 25  
Current State Architecture 11

## D

Data Object 15  
Derived relations 29–30, 34  
    and worst case scenarios 34  
    Derived Dynamic relations 30  
    Derived Structural relations 29  
    limitations of 34  
Device 19–20  
Donate 12  
Dreyfus, Hubert 74

## E

Enterprise Architecture 10  
    Fragmentation 10  
    Layers 10  
Excel  
    as an Application Component 46  
    as System Software 48

## F

Flow relation 22  
Future State Architecture 11

## G

GOFBF 74



## Grouping

- not using the Grouping relation 43
- using the Grouping relation 26

## H

Hacker, P.M.S. 9

## I

- Infrastructure Function 18–19
- Infrastructure Interface 18–19
- Infrastructure Layer 10
- Infrastructure Service 18–19

## J

Junction relation 27

## M

Meaning 25

## N

- Nesting 20–21
- Network 26
- Node 18
  - using for infrastructure encapsulation 21

## P

Pattern

- Anti-Pattern 63
  - Application Collaboration 63
  - Association 65
- Application Platform 49
- Business
  - Business Functions 77
  - End-to-end Business Process 75–76
- Business Pattern
  - Enterprise as loose collaboration of Business Functions 72
- Concurrent Realization 61
- Database Server 44
- Deployment Pattern
  - Classic Two-Tier Application 52–53
  - Providing a local ASP 56–57
  - Remote ASP 53
  - SaaS 55–56
  - Standalone PC 51
  - Standalone PC with Shared Data 51
  - Three-Tier Application 55

Infrastructure Building Blocks 50

Infrastructure Pattern

- Database Cluster 60
- Database Replication 57
- Server Cluster 60

Internet Browser 48

Real World Example 65–69

Secondary and Tertiary Architecture

- Pragmatic set of Roles to add to your model 87
- Spreadsheet 46
- Using collections in a model 60

Patterns 44

Application Deployment 51–56

Print License 12

Product 25

Project Start Architecture. *See* Change Architecture

## R

- Realization relation 16–17, 19
- Representation 25
- Risk 90
- Risk and Security modeling 90–95

## S

- Secondary and Tertiary Architecture 84
- Specialization relation 24
- System Software 19–20
  - versus Application Component 35

## T

Trigger relation 22

## U

- UML 32, 34
- Used-By relation 17, 19

## V

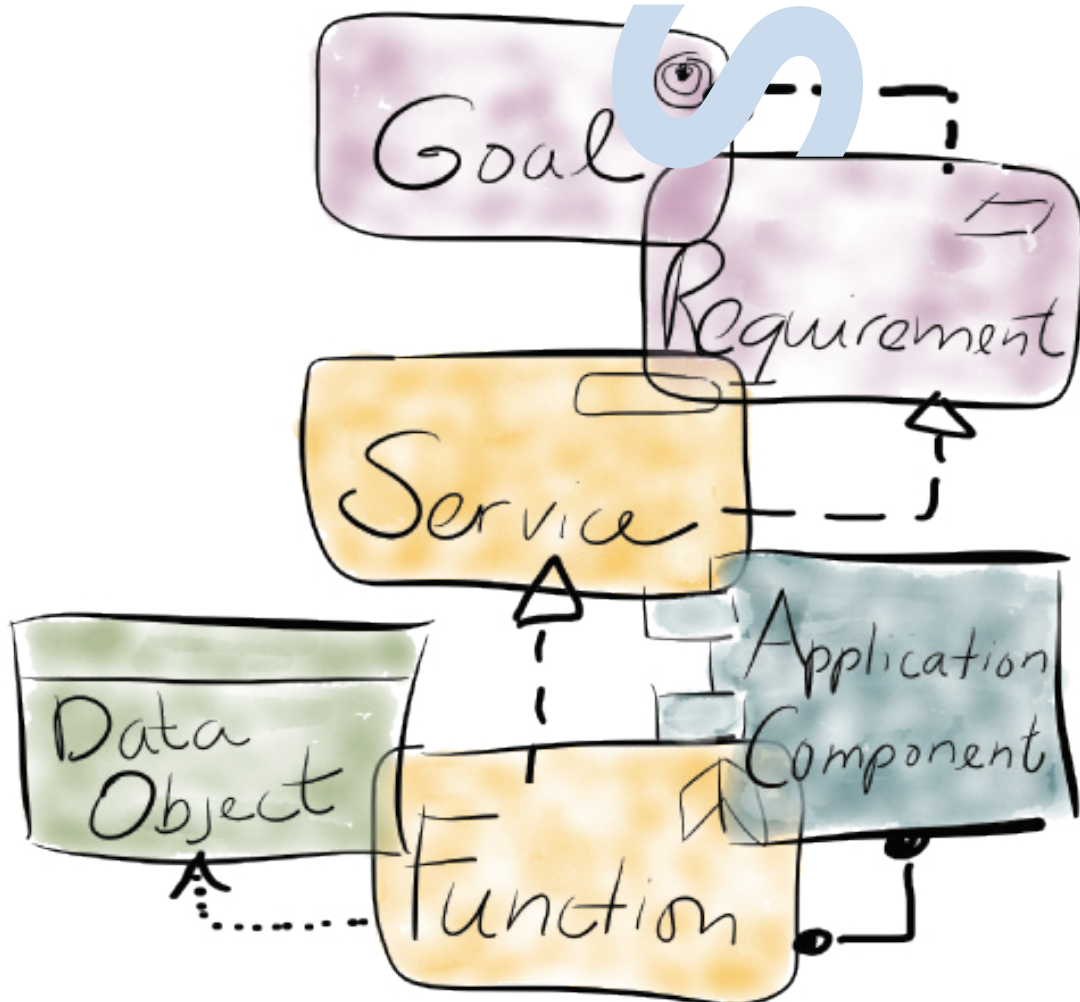
Value 25

## W

Wittgenstein, Ludwig 9, 36, 72, 101, 103, 104, 112, 113

This page intentionally not left blank

# to raise awareness





# List of Figures

View 1. The Basic Application Pattern.....	15	View 27. Representation and Data Object Realizing a Business Object.....	25
View 2. Basic Business Process Pattern .....	16	View 28. Network and Communication Path .....	26
View 3. Basic Application is used by Basic Business Pattern.....	17	View 44. Aggregation can be used as grouping.....	26
View 4. An Application Using Another Application .....	17	View 29. Automated Process .....	26
View 5. Business Actor in Context .....	18	View 30. Grouping Relation .....	27
View 43. Basic Application uses Basic Infrastructure ...	18	View 31. Junction relation (join) .....	27
View 6. Write Answer Process, supported by MS Word and a Standalone PC .....	19	View 32. Junction relation (split).....	27
View 7. Device and System Software.....	20	View 33. Example use of Location object .....	27
View 8. Composition Example.....	20	View 45. ArchiMate 2.0 Metamodel .....	28
View 9. Aggregation Example.....	20	View 46. Write Answer Process, supported by MS Word and a Standalone PC .....	29
View 10. Nested Aggregation .....	20	View 34. The Basic Application Pattern.....	30
View 11. Two Nested Aggregations with Shared Object	21	View 35. Derived Dynamic Relations.....	30
View 12. Assignment Nesting, Two Levels Deep .....	21	View 36. Artifact Used-By Application Service? .....	30
View 13. Device and System Software Nested in a Node. 21		View 37. Artifact Used-By a Business Service: the hidden objects.....	31
View 14. Device, System Software and Infrastructure Function Nested in a Node .....	21	View 47. Not quite what ArchiMate intended, .....	31
View 15. Trigger and Flow .....	22	View 38. This was possible in ArchiMate 1.0 .....	31
View 16. Trigger, Flow and Access to Objects.....	22	View 39. How Function and Service are displayed in the Meta-model.....	32
View 17. Business Functions sharing Access to a Business Object, without Flows .....	22	View 40. How the function/service relations in the meta-model are intended. ....	32
View 18. Business Functions sharing Access to a Business Object, with Flows .....	22	View 41. Example used for discussing ‘worst case’ versus ‘happy flow’ assumptions about dependencies .....	34
View 19. Business Event Triggers Business Process.....	22	View 42. From Node to Business Layer.....	35
View 20. Collaboration: Sales Uses Legal on Order Creation.....	23	View 76. Style: Overlapping Lines .....	39
View 21. Collaboration: Sales and Legal Collaborate on Order Creation .....	23	View 48. Style: non-overlapping lines.....	40
View 22. Collaboration: Quick and Dirty Method of modeling .....	24	View 77. ArchiMate Metamodel in Bad Layout .....	40
View 23. Application Collaboration.....	24	View 49. Style: Non-aligned Relations.....	41
View 24. Specialization Example: Investment Types.....	24	View 50. Style: Aligned and Equally Sized objects, Aligned Relations .....	41
View 25. Specialization Example: Multiple Inheritance	24	View 51. Style: Equally-sized, non-aligned objects .....	41
View 26. Product, Contract, Value and a Product’s Constituents .....	25	View 52. Style: Variable-sized, non-aligned objects .....	41
		View 53. Style: Variable-sized, non-aligned and non-ordered objects.....	41
		View 54. Style: Align nested objects .....	42

View 55. Style: Objects not distributed evenly in their 'group'.....	42
View 56. Full details of deploying a database .....	44
View 78. Deploying a database, without the hardware details.....	45
View 79. Deploying a database, without hardware details and interfaces.....	45
View 81. Deploying a database: using a Node to make an abstract device .....	45
View 80. Deploying a database, without hardware details, interfaces and internal behavior.....	45
View 57. Deploying a database: Nesting details in a Node	46
View 58. Deploying a database: Collapsing the Node .	46
View 82. Excel as an Application Component .....	46
View 59. Excel as an Application: linking a plugin to the Application Service .....	47
View 60. Excel and the plugin as an Application Collaboration.....	47
View 83. Excel as Infrastructure Service (Application Platform) .....	48
View 61. The Internet Browser as Infrastructure Service (Application Platform).....	49
View 84. Database as an Application Platform .....	49
View 85. BRE as Application Platform .....	50
View 86. TI Building Blocks.....	50
View 62. Deployment Pattern: Standalone PC .....	51
View 63. Deployment PatternL Standalone PC using a File Share.....	51
View 64. Deployment Pattern: Classic Two-Tier Application.....	52
View 87. Deployment Pattern: Two-Tier Application with mounted client and two databases.....	53
View 65. Deriving the relations from Infrastructure Service to Infrastructure Service (Used-By) and from Node to Infrastructure Service (Realization).....	53
View 88. Deployment Pattern: Three-tier Application (with missing business logic) .....	54
View 89. Deployment Pattern: Three-tier Application...	54
View 90. Deployment Pattern: Two-Tier Application with remote server.....	55
View 91. Deployment Pattern: Software as a Service (SaaS) - variation 1 .....	56
View 92. Deployment Pattern: SaaS Provider uses our Infrastructure .....	56
View 93. Providing your own ASP .....	57
View 94. Providing a local ASP with all the details.....	58
View 95. Infrastructure Pattern: Database Replication..	58
View 96. Infrastructure Pattern: Database Cluster. Set up pioneered by colleague/hired gun Roy Zautsen .....	59
View 97. Infrastructure Pattern: High Available Server. Set up pioneered by colleague/hired gun Roy Zautsen. ....	60
View 66. Adding collections to your model by Specializa-	61

View 67. Adding collections to your model using Aggrega-	61
View 68. Two Business Functions Concurrently Realize One Business Service .....	61
View 69. Two Functions Interact to Realize a Service...	61
View 70. Two Business Functions each Realize a Sub-Ser-	62
vice of an Overall Business Service .....	62
View 71. Two Business Functions each Realize a Sub-Ser-	62
vice of an Overall Business Service, expanded .....	62
View 99. Application C uses Application A and Applica-	62
tion B .....	62
View 98. Application C uses Application A which uses	62
Application B .....	62
View 72. Two independent (Sub-)Services make up a sin-	62
gle Product.....	62
View 73. Application Service Realized by Two Application	62
Functions.....	62
View 74. Collaboration: Sales Uses Legal on Order Cre-	63
ation.....	63
View 75. Collaboration: Sales and Legal Collaborate on	63
Order Creation .....	63
View 100. Application C uses a Collaboration of Applica-	63
tions A and B.....	63
View 102. Using Application Collaboration to model a	64
multi-tiered software system .....	64
View 101. Application C uses a Collaboration of Applica-	64
tions A and B, with Application Interface.....	64
View 103. Modeling the Use of a Managed File Transfer	66
and a Scheduler use for your business, a human user loads	66
the retrieved data.....	66
View 104. Modeling the Use of a Managed File Transfer	67
and a Scheduler use for your business, fully automated	67
loading.....	67
View 105. Modeling the Use of a Managed File Transfer	68
and a Scheduler use for your business, fully automated	68
loading (cleaned up).....	68
View 106. Scheduling example: Scheduler triggers ETL	69
View 107. Scheduling example: Scheduler uses ETL....	69
View 108. Nesting a behavioral object inside an active	71
object to suggest encapsulation .....	71
View 109. Encapsulation of Application Sub-Components	72
and Application Sub-Functions .....	72
View 110. Pre-ArchiMate 1.0 relations between Business	72
Process, Business Function and Business Activity.....	72
View 111. A Business Process Aggregates Business Func-	73
tions and a Business Function Aggregates Business Pro-	73
cesses.....	73
View 116. A Business Function's internal process Realizes	73
a Business Service .....	73
View 117. A Business Function's internal process Realizes	73
the Business Function's service, nested view.....	73
View 118. A Business Process using Business Functions is	74
performed by a (loose) Business Collaboration .....	74
View 119. An ArchiMate Business Function, with two	74
internal processes, roles and services realized .....	74

View 120. An ArchiMate Business Function, with two internal processes (nested), roles and services realized	74	View 141. The Primary Architecture with the Secondary Application Management Architecture.....	86
View 121. An ArchiMate Business Function realizing two services with internal processes omitted from View 120	74	View 142. The Primary and Secondary Architectures and the (Tertiary) Application Owner Architecture.....	87
View 122. A Good-Old-Fashioned-Business-Function becomes a Business Role in ArchiMate, further simplified from View 121 .....	74	View 143. The Full Picture: Primary, Secondary and Tertiary Architecture.....	88
View 123. How the process modeler looks at the 'end-to-end' process in a 'swim lanes' kind of view .....	75	View 144. Simplified way to model roles from the Primary/Secondary/Tertiary Architecture Analysis .....	89
View 124. Business Functions added to the process modeler's View 123, the GOFBF way .....	75	View 115. ArchiMate 2.0 Motivation Extension example	91
View 125. The process modeler's View 123, with functions added the (kind of) ArchiMate way .....	75	View 145. Five ways (patterns) the Enterprise Architecture can Realize Control Measures/Controls that contribute to Control Objectives that are associated with Risks .....	93
View 126. The end-to-end process as a chain of processes, without using collaboration and interaction.....	76	View 146. Using Excel for Data Entry for a Business Process .....	94
View 127. End-to-end process modeled as Business Interaction (with added Aggregate relations to the subprocesses) .....	76	View 147. The 'Create Spreadsheet' Excel Application Function.....	94
View 128. The end-to-end process modeled as a Business Interaction, strictly following the description of Business Interaction in the ArchiMate 2.0 specification.....	76	View 148. Large Model Construction - Business Architecture View .....	97
View 129. Simplified Portfolio Management without Business Functions .....	77	View 149. Large Model Construction - Application Architecture View .....	97
View 130. Simplified Portfolio Management with Ex-Ante Compliance as part of Portfolio Management function	78	View 150. ArchiMate 2.0 Construction Views .....	98
View 131. Simplified Portfolio Management with Ex-Ante Compliance as part of Compliance Control function ...	78	View 151. ArchiMate 2.0 Use Views: Application Exploitation and Risk& Security .....	99
View 132. Simplified Portfolio Management with Ex-Ante Compliance as part of the Compliance Control Function	78	View 152. Large Model Construction – Infrastructure Architecture View .....	100
View 133. Simplified Portfolio Management with Ex-Ante Compliance as part of the Portfolio Management function	79	View 155. Large Model Use – Risk & Security View...	100
View 134. Simplified Portfolio Management with Ex-Ante Compliance part of the Compliance Control function but performed by the Portfolio Manager role .....	79	View 154. Large Model Use – The Run Manager's Application View .....	100
View 135. Simplified Portfolio Management with Ex-Ante Compliance part of the Portfolio Management function but performed by the Compliance Officer role .....	80	View 153. Large Model Construction – Data-Information View.....	100
View 136. Simplified Portfolio Management with the Compliance Control function performed by a Collaboration of Portfolio Manager and Compliance Officer.....	80	View 156. The Basic Application Pattern.....	103
View 137. Simplified Portfolio Management with the Portfolio Management function performed by a Collaboration of Portfolio Manager and Compliance Officer.....	81	View 157. Proposal: A service is the Composite usable part of a Function .....	104
View 112. Cash & Payments function and Accounting Function both have a reconciliation process.....	82	View 158. Human processes: Business Objects are Realized, applications are Used .....	105
View 113. A separate Reconciliation Business Function	82	View 159. Automated Process: the Application Component is Assigned-To (performs) the Business Process and the Application Interface is Assigned-To (is the 'protocol' for) the Business Service .....	105
View 114. A separate Reconciliation Business Function still uses the accounting and payments systems for exception handling.....	82	View 160. An automated process with human exception handling.....	105
View 139. The Primary Architecture: Business uses an Application that uses Infrastructure.....	84	View 161. When two roles separately do the (sub) Business Processes, the overall Business Process is performed by a Collaboration.....	106
View 138. The Primary Architecture with the Secondary Development Architecture.....	84	View 162. Proposed meta-model Realization relations for automated processes .....	106
View 140. The Primary Architecture with the Secondary Exploitation Architecture .....	85	View 163. Proposed solution for non-automated processes	106
		View 169. An example that uses the proposed changes to ArchiMate from sections 17.1, 17.2 and 17.3. A Business Process is performed by a Collaboration of humans and applications. The process is automated using a BRE, humans handling the exceptions, using a user interface of the BRE and some hand work.....	107
		View 164. A Business Role as Aggregate child of a Business Actor.....	108



View 170. Customer Support and Sales Work together to organize the yearly office party..... 108

View 165. Customer Support and Sales both have the responsibility (Role) for organizing the yearly office party 108

View 171. Artifact as Composite part of Device..... 109

View 172. Artifact as Composite part (derived relation) of a Node..... 109

View 173. Exploitation Infrastructure Services based on Aggregation of (shared) Infrastructure Service Building Blocks ..... 110

View 174. Exploitation Infrastructure Service based being a Composition child of Multiple parents ..... 110

View 166. Three useful (derived) Access relations ..... 111

View 167. An Infrastructure Service Used-By an Artifact? 111

View 168. How the Infrastructure Function can be (Used-By (derived) by a passive Artifact in ArchiMate ..... 112

View 175. ArchiMate 2.0 Metamodel. for easy reference. 131

This page intentionally not left blank

# የግንባታ የጥራት





# Finger

This Chapter contains material for easy while reading the book You keep one finger between this page and the next as the next page has the ArchiMate meta-model.

Additionally: here is an overview of the Style Guide Items:

## Style Guidelines

1. Make your relations in general go in vertical and horizontal directions only.
2. Don't let relations overlap
3. Minimize the number of line crossings
4. As much as possible: Group relations according to either source or destination
5. Align relations, even unrelated ones
6. Align objects, even unrelated ones.
7. Use as few as possible different object sizes (this is like not using too many font sizes in a text document).
8. Align objects and attach relations such that relations are as simple as possible and with the least number of line crossings, preferably straight lines from one object to another.
9. If you have a nested object or groupings, align the objects that are on the inside as well.
10. Distribute objects evenly within their 'group'.
11. *Make a view as easy on the eye, as 'quiet' as possible without losing essential information.*

This page intentionally not left blank





**Mastering ArchiMate** is a book about the ArchiMate® Enterprise Architecture Modeling Language, which is an open standard and a Registered Trade Mark of The Open Group. This book gives an introduction to the language and then goes on to show you many different patterns for use. From Business to Infrastructure, from Risk & Security to Application Exploitation and Maintenance.

**Gerben Wierda (1961)** is Lead Architect of APG Asset Management, one of the largest Fiduciary Managers in the world. He has overseen the construction of one of the largest single ArchiMate models in the world to date. He holds an M.Sc. in Physics from the University of Groningen and an MBA from RSM Erasmus, Rotterdam.

© Gerben Wierda, 2012

[info@masteringarchimate.com](mailto:info@masteringarchimate.com)

ISBN 978-90-819840-0-3

Price: see "License" on page 12

ISBN 978-90-819840-0-3



9 789081 984003